Computer Graphics

8 - Hierarchical Modeling

Yoonsang Lee
Spring 2021

Topics Covered

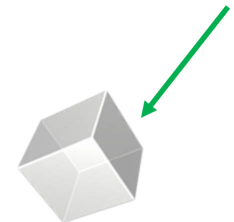
- Meanings of an Affine Transformation Matrix
- Interpretation of a Series of Transformations
- Hierarchical Modeling
  - Concept of Hierarchical Modeling
  - OpenGL Matrix Stack

Meanings of an Affine Transformation Matrix

Meanings of an Affine Transformation Matrix

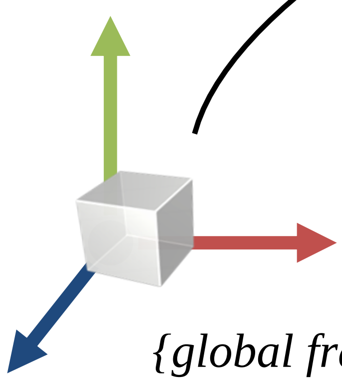
- To understand hierarchical modeling, let's first take a closer look at the meaning of an affine transformation matrix.

1) A 4x4 Affine Transformation Matrix transforms a Geometry w.r.t. Global Frame



Translate, rotate, scale, ...

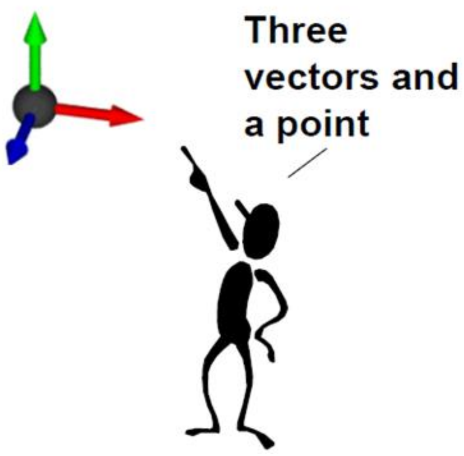
$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Every vertex position (w.r.t. the global frame) of the cube is transformed to another position (w.r.t. the global frame)

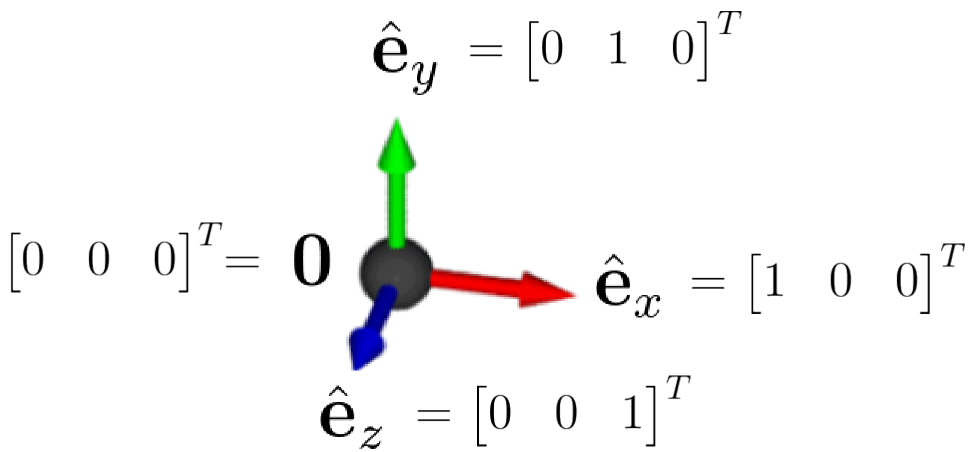
Review: Affine Frame

- An **affine frame** in 3D space is defined by three vectors and one point
  - Three vectors for x, y, z axes
  - One point for origin



Global Frame

- A **global frame** is usually represented by
  - Standard basis vectors for axes : $\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z$
  - Origin point : $\mathbf{0}$



Let's transform a "global frame"

- Apply M to this "global frame", that is,
  - Multiply M with the x, y, z axis *vectors* and the origin *point* of the global frame:

x axis *vector*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \\ \boxed{0} \end{bmatrix}$$

y axis *vector*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} m_{12} \\ m_{22} \\ m_{32} \\ \boxed{0} \end{bmatrix}$$

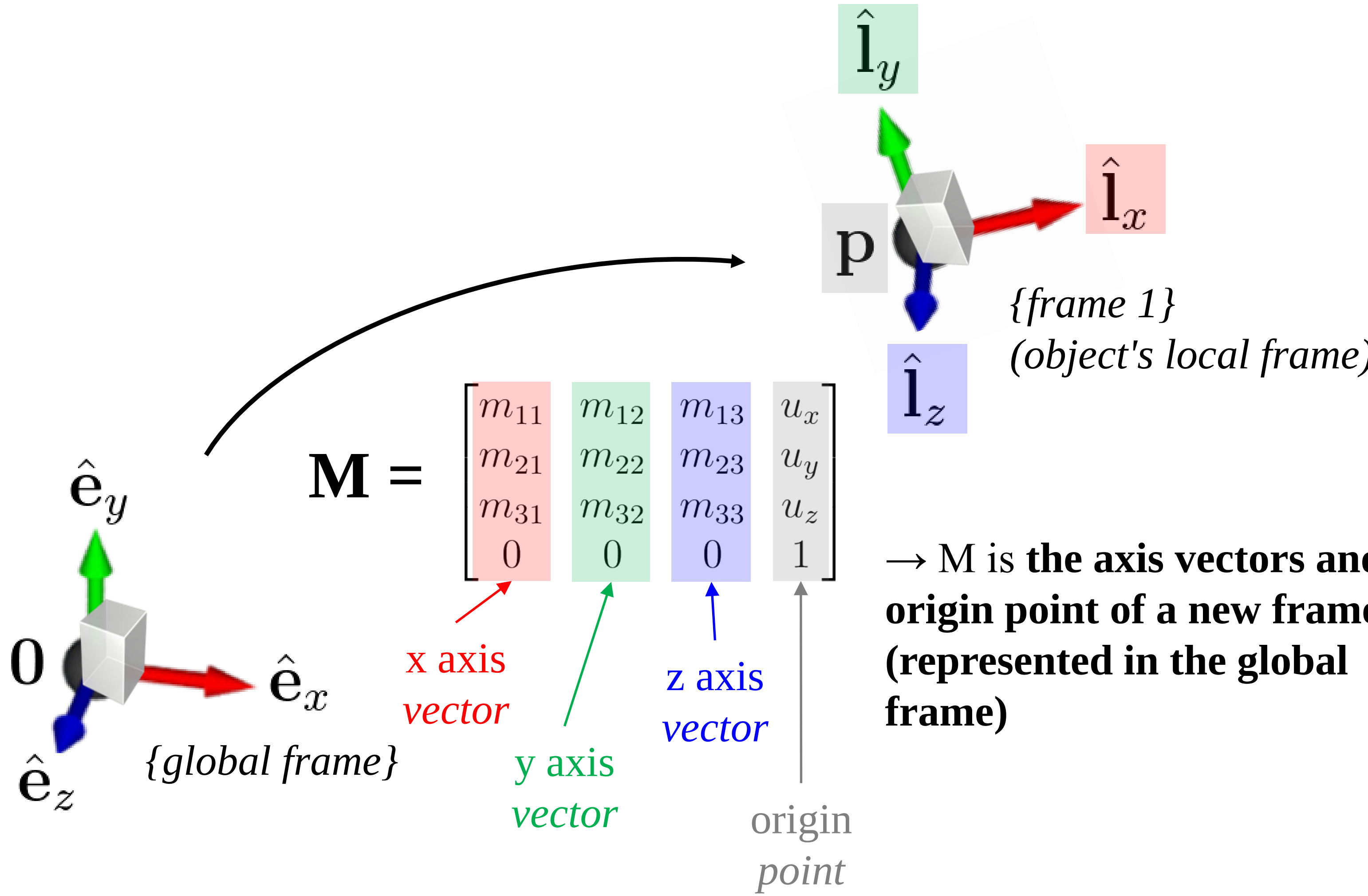
z axis *vector*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} m_{13} \\ m_{23} \\ m_{33} \\ \boxed{0} \end{bmatrix}$$

origin *point*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \boxed{1} \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \\ u_z \\ \boxed{1} \end{bmatrix}$$

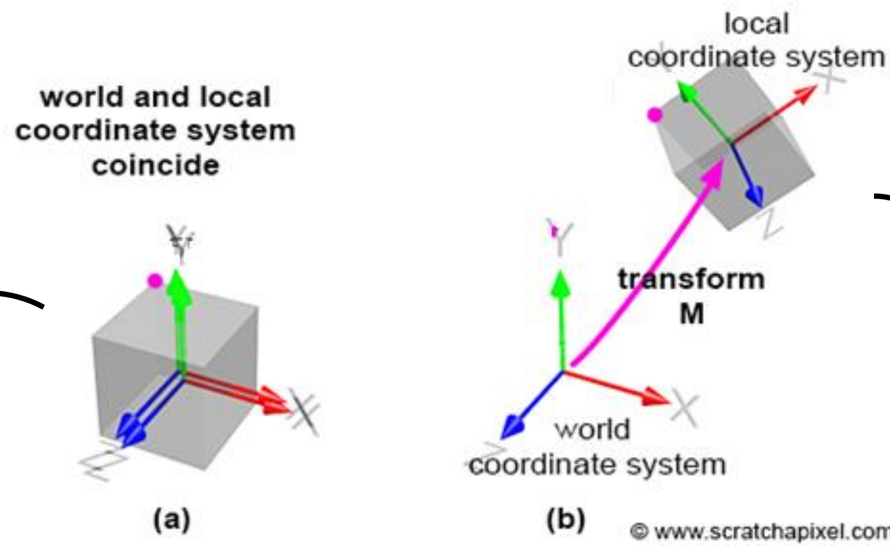
2) A 4x4 Affine Transformation Matrix defines an Affine Frame w.r.t. Global Frame



$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

→ M is **the axis vectors and origin point of a new frame (represented in the global frame)**

Examples



The object's local frame is defined by:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x axis *vector* y axis *vector* z axis *vector* origin *point*

of the local frame ***represented in the global frame***

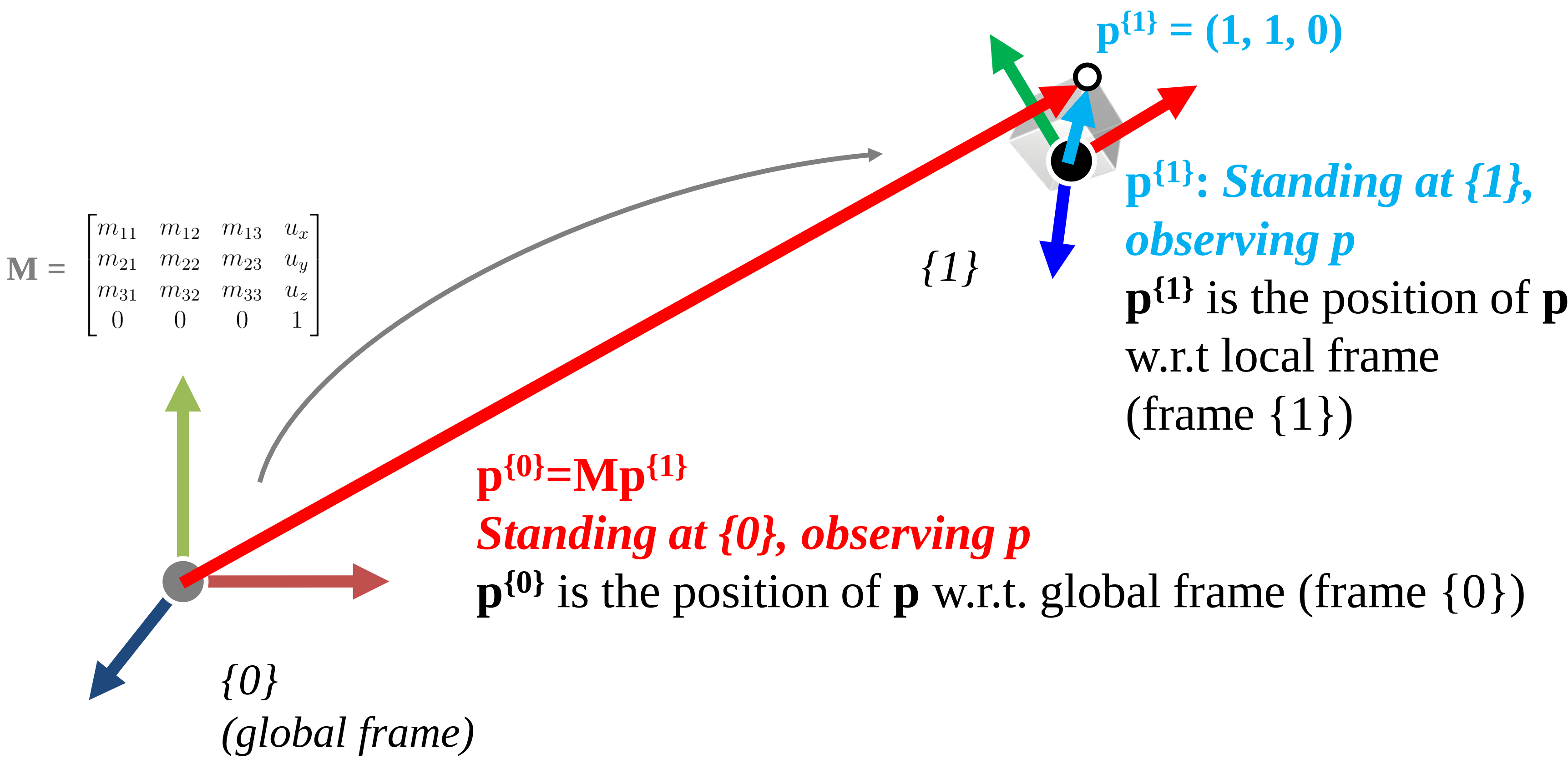
The object's local frame is defined by:

origin *point*

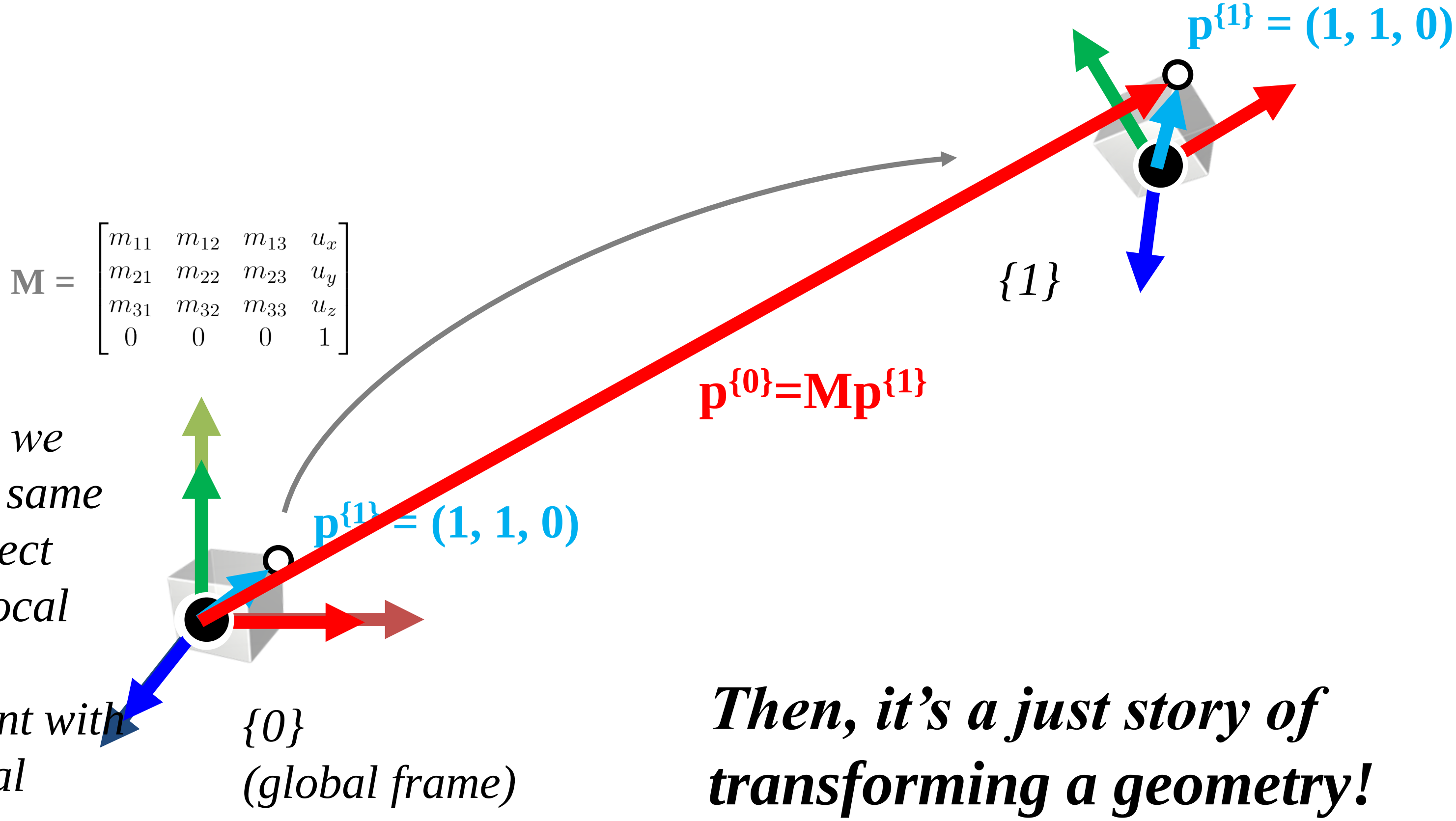
$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_1 \\ m_{21} & m_{22} & m_{23} & u_2 \\ m_{31} & m_{32} & m_{33} & u_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x axis *vector* y axis *vector* z axis *vector*

3) A 4x4 Affine Transformation Matrix transforms a Point Represented in an Affine Frame to (the same) Point (but) Represented in Global Frame



3) A 4x4 Affine Transformation Matrix transforms a Point Represented in an Affine Frame to (the same) Point (but) Represented in Global Frame Because...



Let's say we have the same cube object and its local frame coincident with the global frame

Then, it's a just story of transforming a geometry!

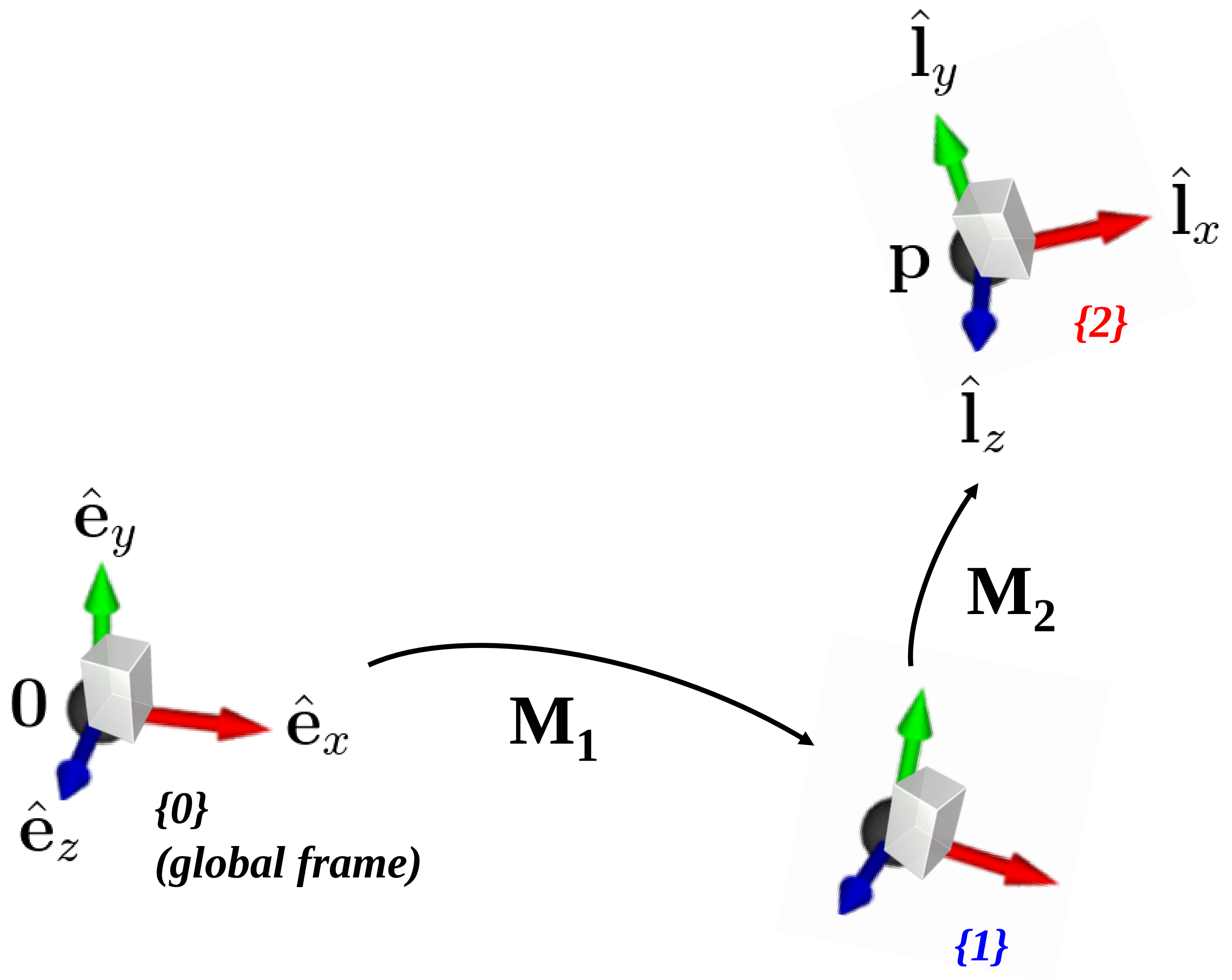
Quiz #1

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

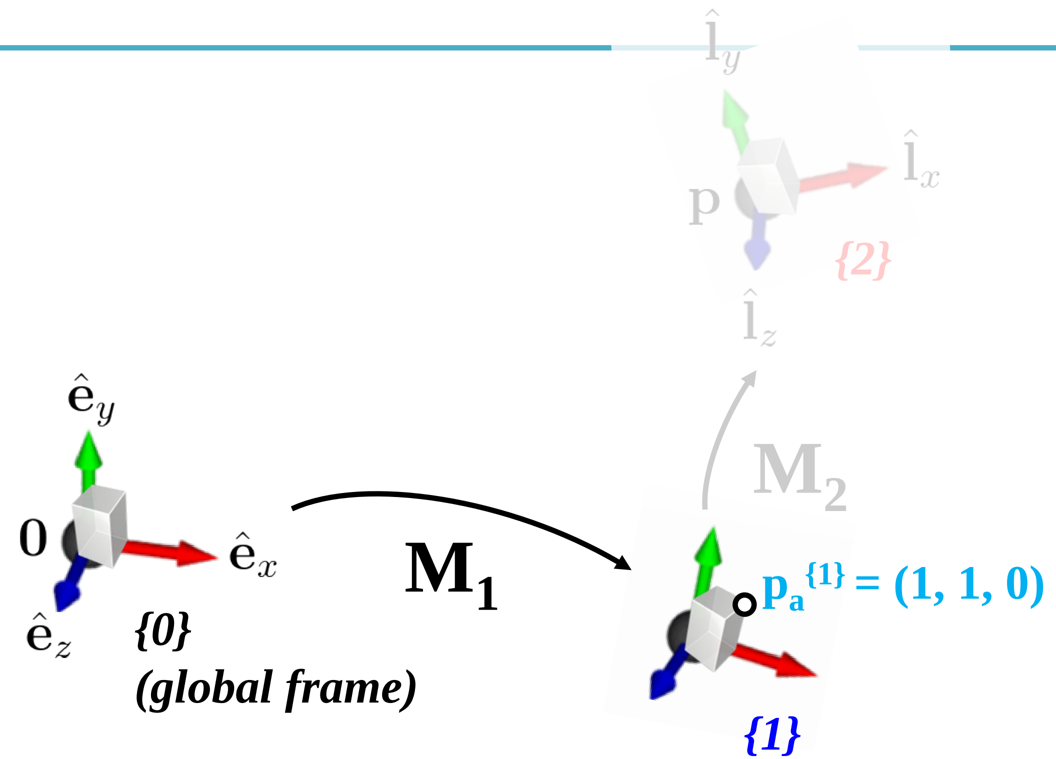
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

All these concepts works even if the starting frame is not global frame!

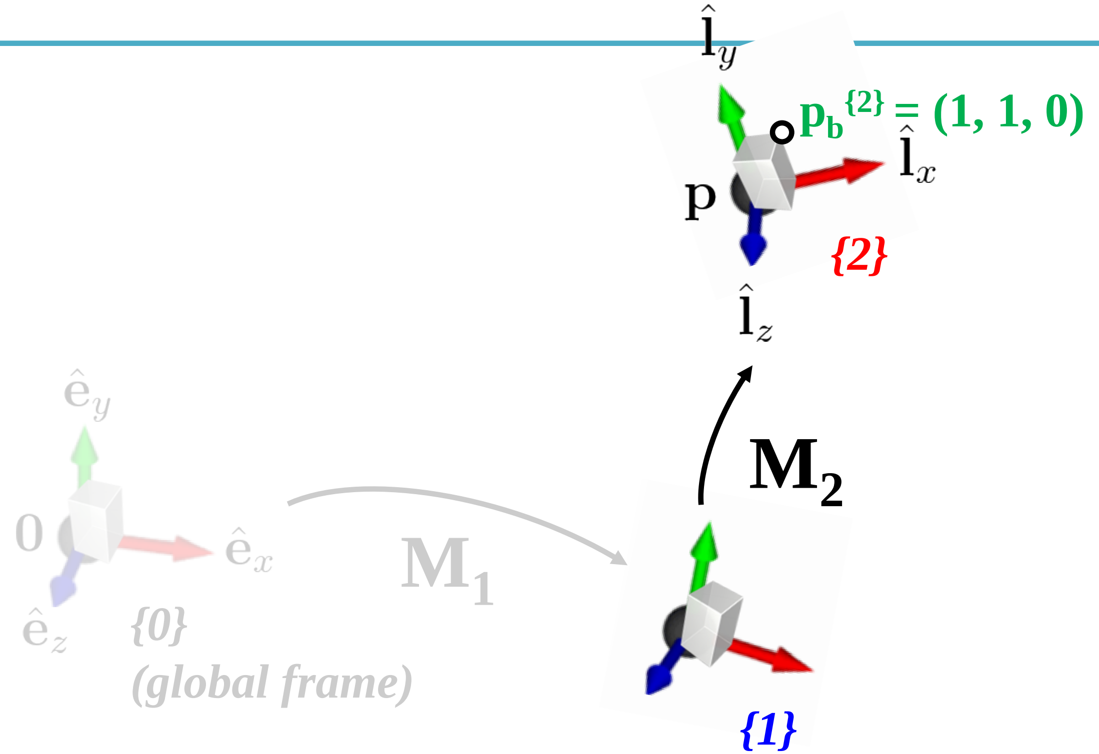


{0} to {1}



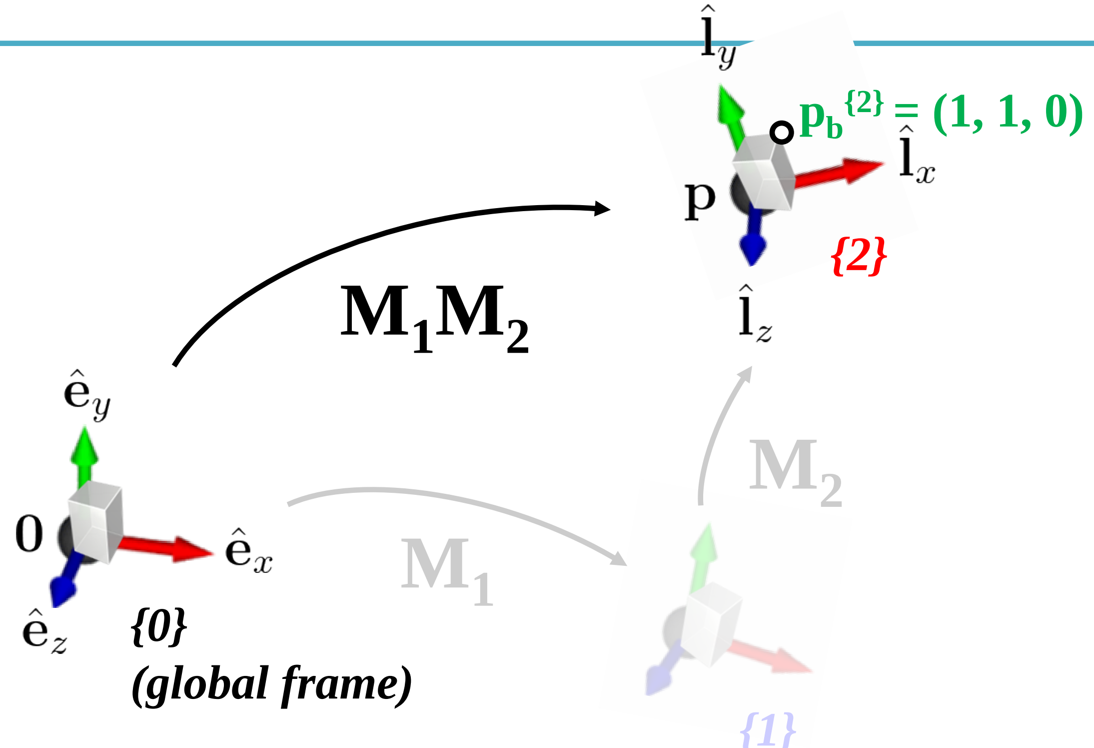
- 1) $\mathbf{M}_1$ transforms a geometry (represented in ***{0}***) w.r.t. ***{0}***
- 2) $\mathbf{M}_1$ defines an ***{1}*** w.r.t. ***{0}***
- 3) $\mathbf{M}_1$ transforms a point represented in ***{1}*** to the same point but represented in ***{0}***
  - $\mathbf{p}_a^{\{0\}} = \mathbf{M}_1 \mathbf{p}_a^{\{1\}}$

{1} to {2}



- 1) $\mathbf{M}_2$ transforms a geometry (represented in *{1}*) w.r.t. *{1}*
- 2) $\mathbf{M}_2$ defines an *{2}* w.r.t. *{1}*
- 3) $\mathbf{M}_2$ transforms a point represented in *{2}* to the same point but represented in *{1}*
  - $\mathbf{p_b}^{\{1\}}=\mathbf{M}_2\mathbf{p_b}^{\{2\}}$

{0} to {2}



- 1) $\mathbf{M_1M_2}$ transforms a geometry (represented in ***{0}***) w.r.t. ***{0}***
- 2) $\mathbf{M_1M_2}$ defines an ***{2}*** w.r.t. ***{0}***
- 3) $\mathbf{M_1M_2}$ transforms a point represented in ***{2}*** to the same point but represented in ***{0}***
  - $\mathbf{p_b}^{\{1\}}=\mathbf{M_2 p_b}^{\{2\}}$, $\mathbf{p_b}^{\{0\}}=\mathbf{M_1 p_b}^{\{1\}}=\mathbf{M_1M_2p_b}^{\{2\}}$

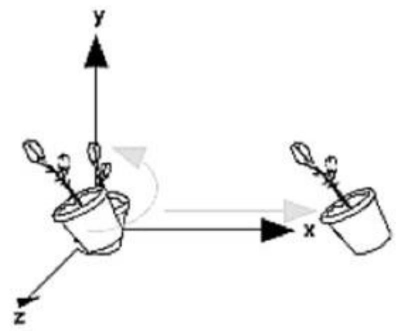
Interpretation of a Series of Transformations

Revisit: Order Matters!

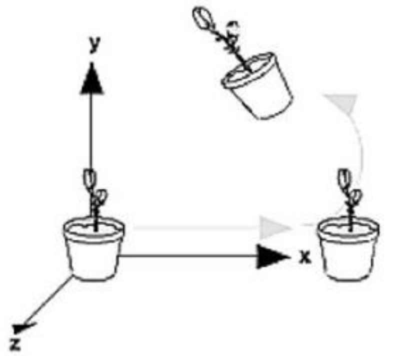
- If T and R are matrices representing affine transformations,

- **p'** = TR**p**
  - First apply transformation R to point **p**, then apply transformation T to transformed point R**p**

- **p'** = RT**p**
  - First apply transformation T to point **p**, then apply transformation R to transformed point T**p**



Rotate then Translate



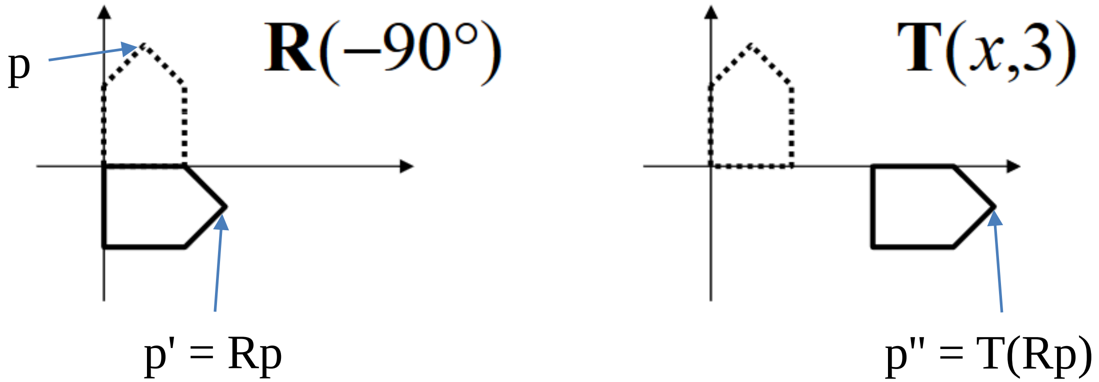
Translate then Rotate

Interpretation of Composite Transformations #1

- An example transformation:

$$M = \mathbf{T}(x,3) \cdot \mathbf{R}(-90°)$$

- This is how we've interpreted so far:
  - R-to-L: Transforms ***w.r.t. global frame***

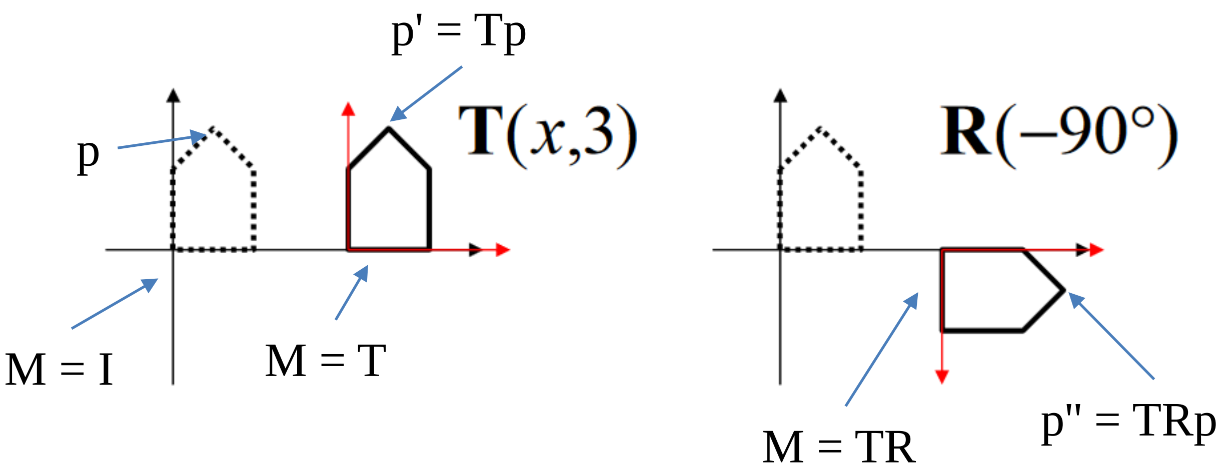


Interpretation of Composite Transformations #2

- An example transformation:

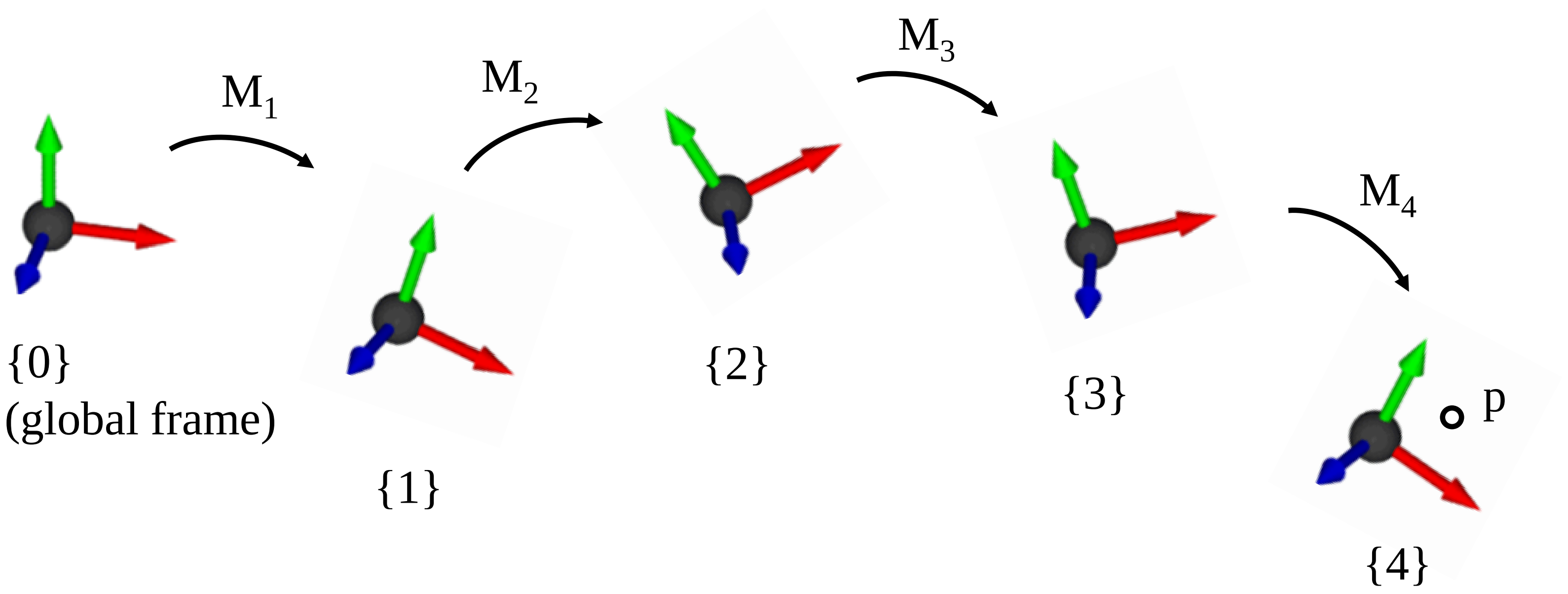
$$M = \mathbf{T}(x,3) \cdot \mathbf{R}(-90°)$$

- **Another way of interpretation:**
  - L-to-R: Transforms ***w.r.t. local frame***



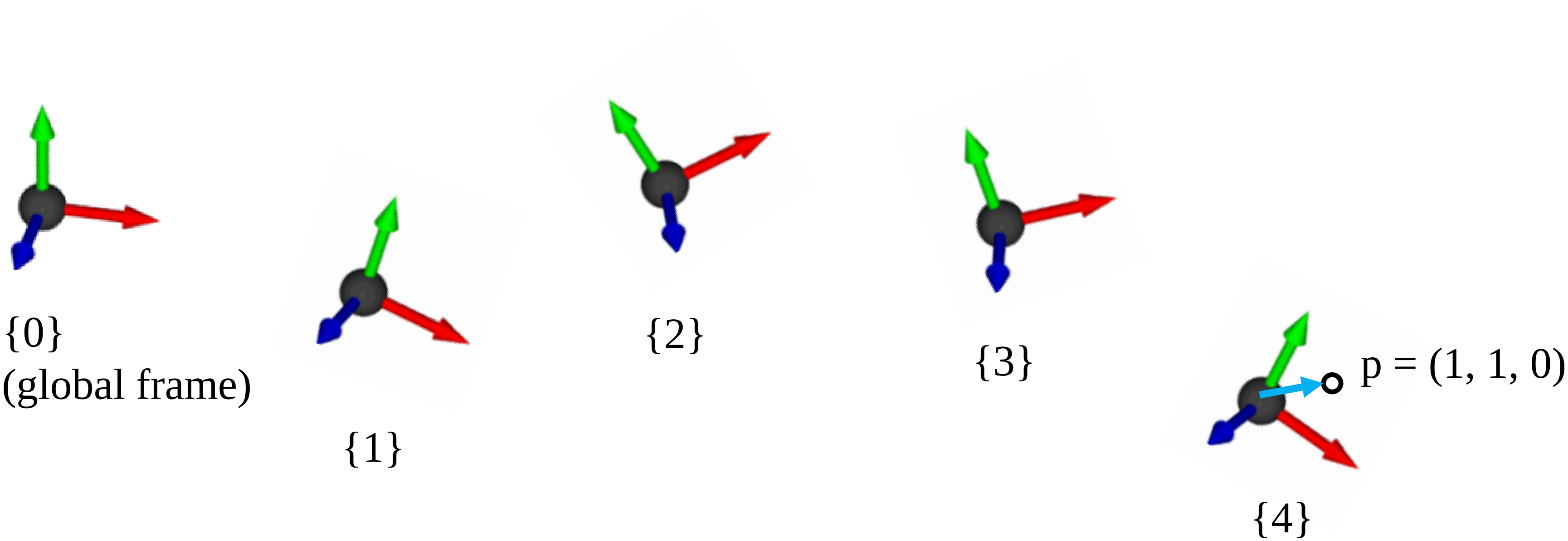
Interpretation of a Series of Transformations #1

- $p' = M_1 M_2 M_3 M_4\ p$



Interpretation of a Series of Transformations #1

- $p' = M_1M_2M_3M_4\boxed{p}$

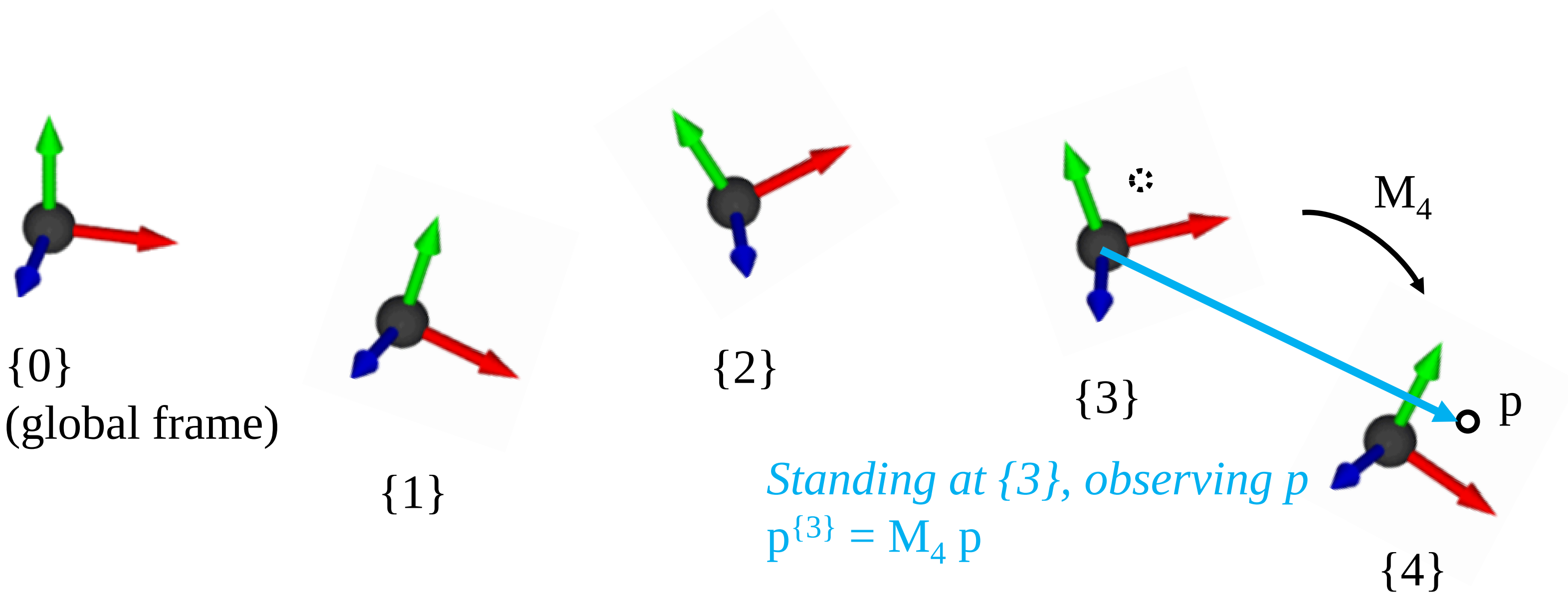


Standing at {4}, observing p

$p^{\{4\}} = p$

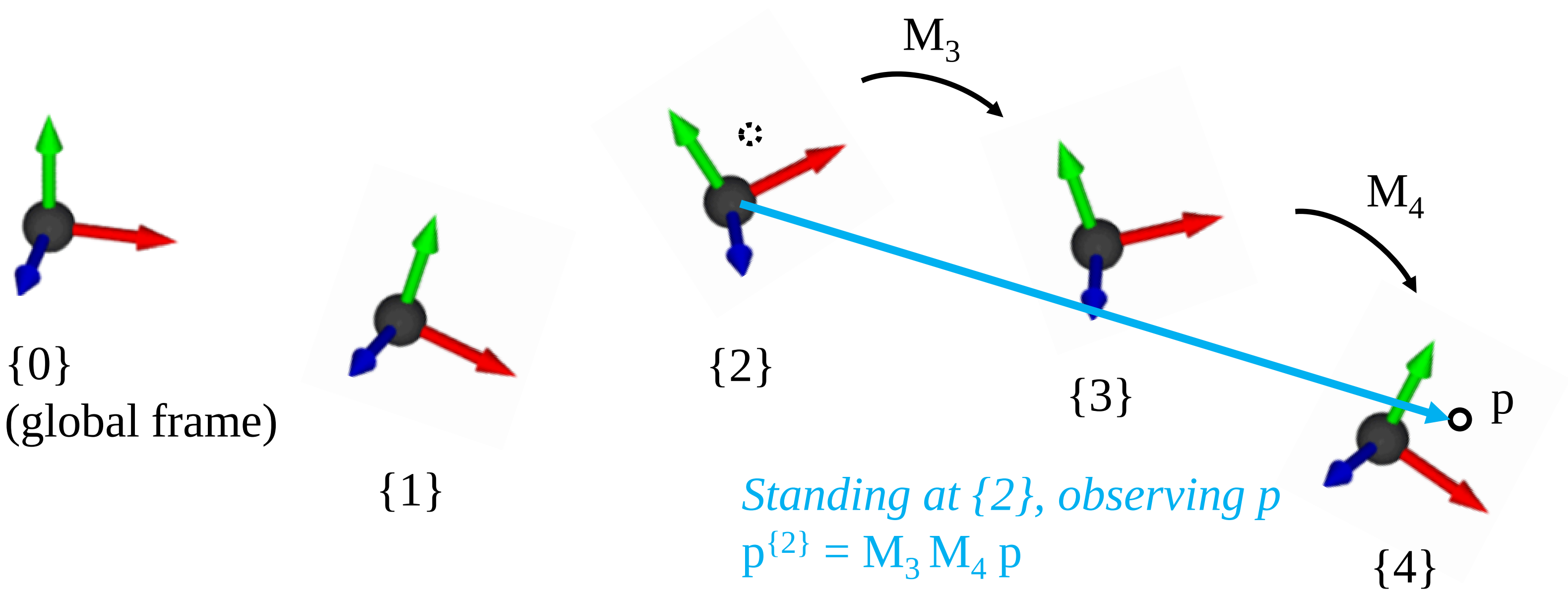
Interpretation of a Series of Transformations #1

- $p' = M_1 M_2 M_3 \boxed{M_4\ p}$



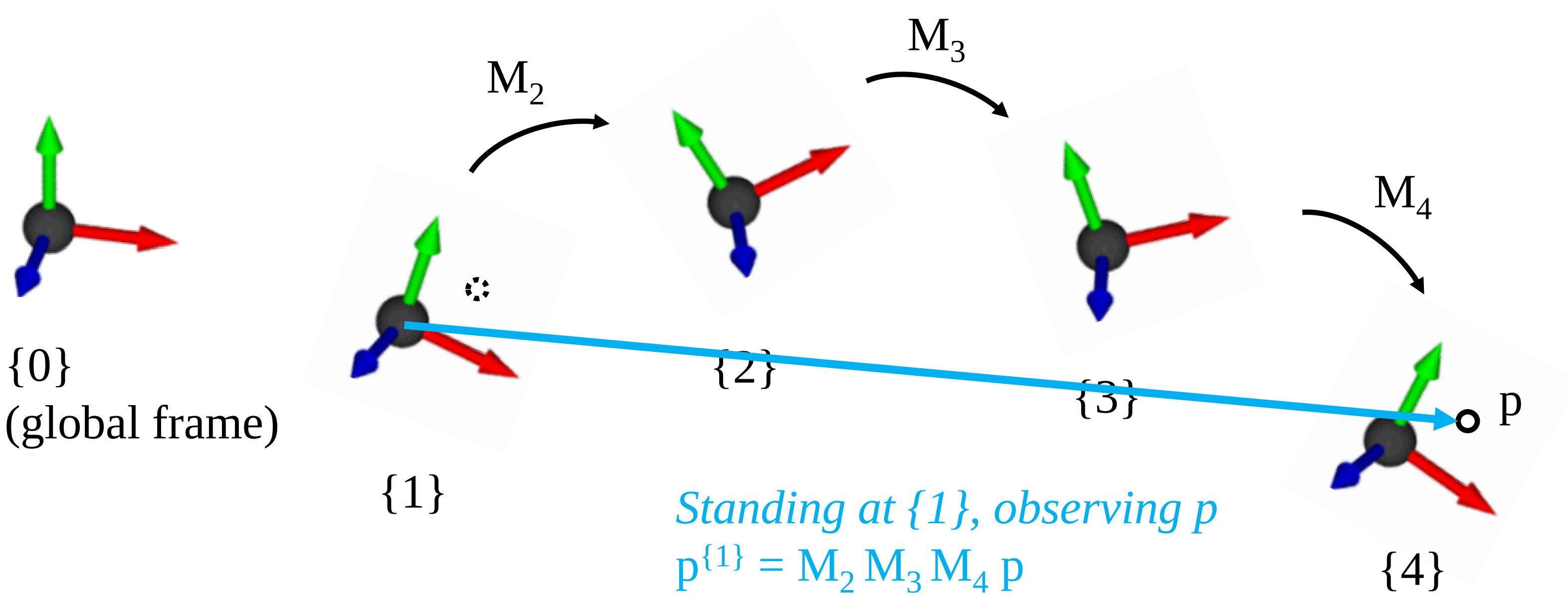
Interpretation of a Series of Transformations #1

- $p' = M_1 M_2 \boxed{M_3 M_4\ p}$



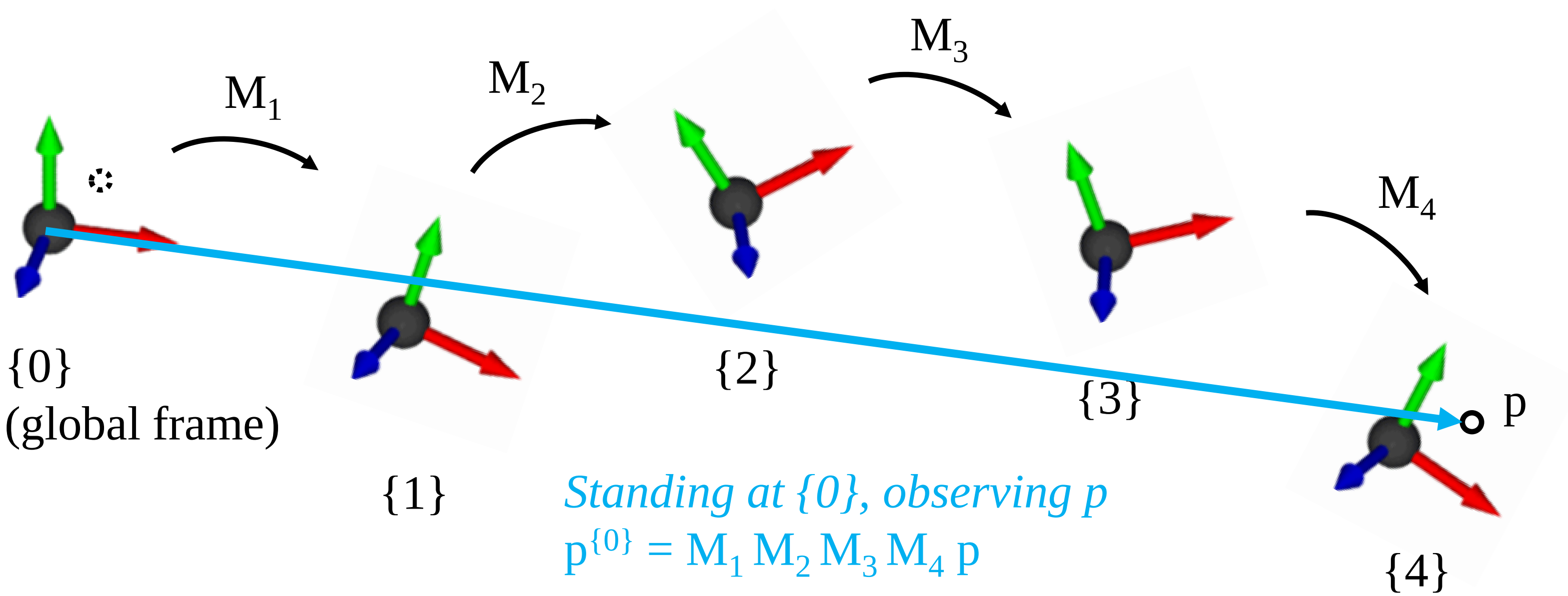
Interpretation of a Series of Transformations #1

- $p' = M_1 \boxed{M_2 M_3 M_4\ p}$



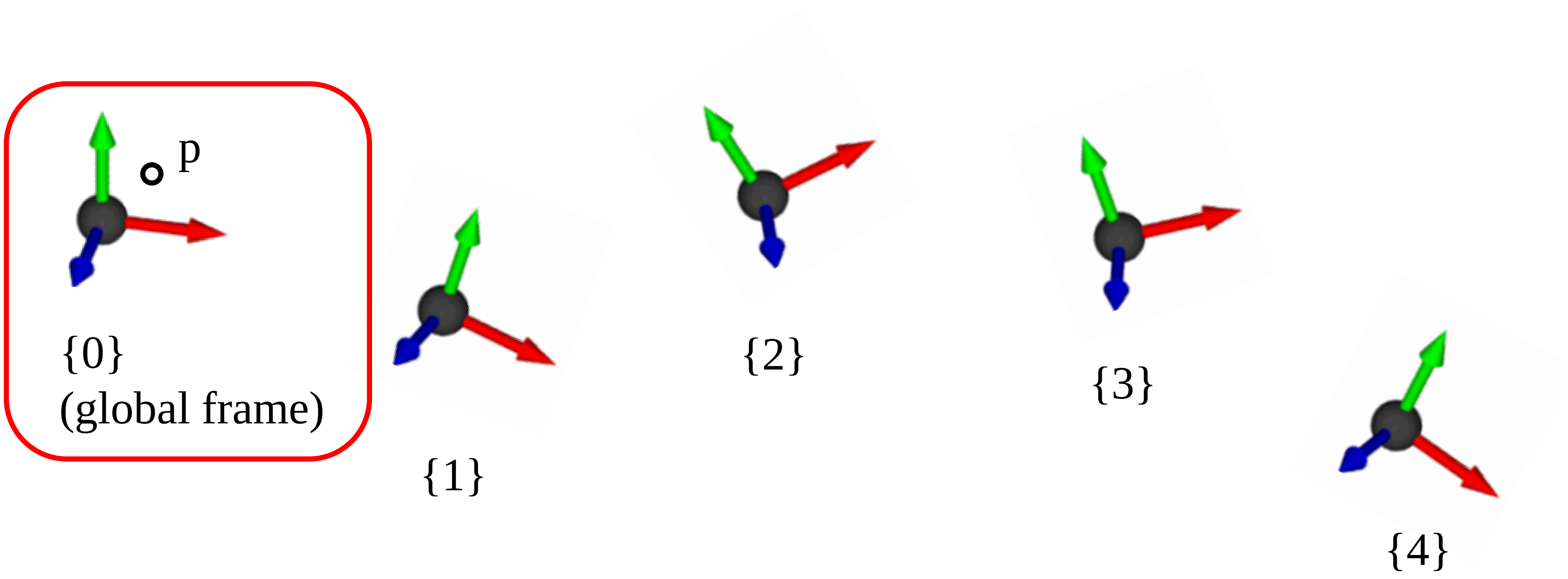
Interpretation of a Series of Transformations #1

- p' = $\boxed{M_1 M_2 M_3 M_4\ p}$



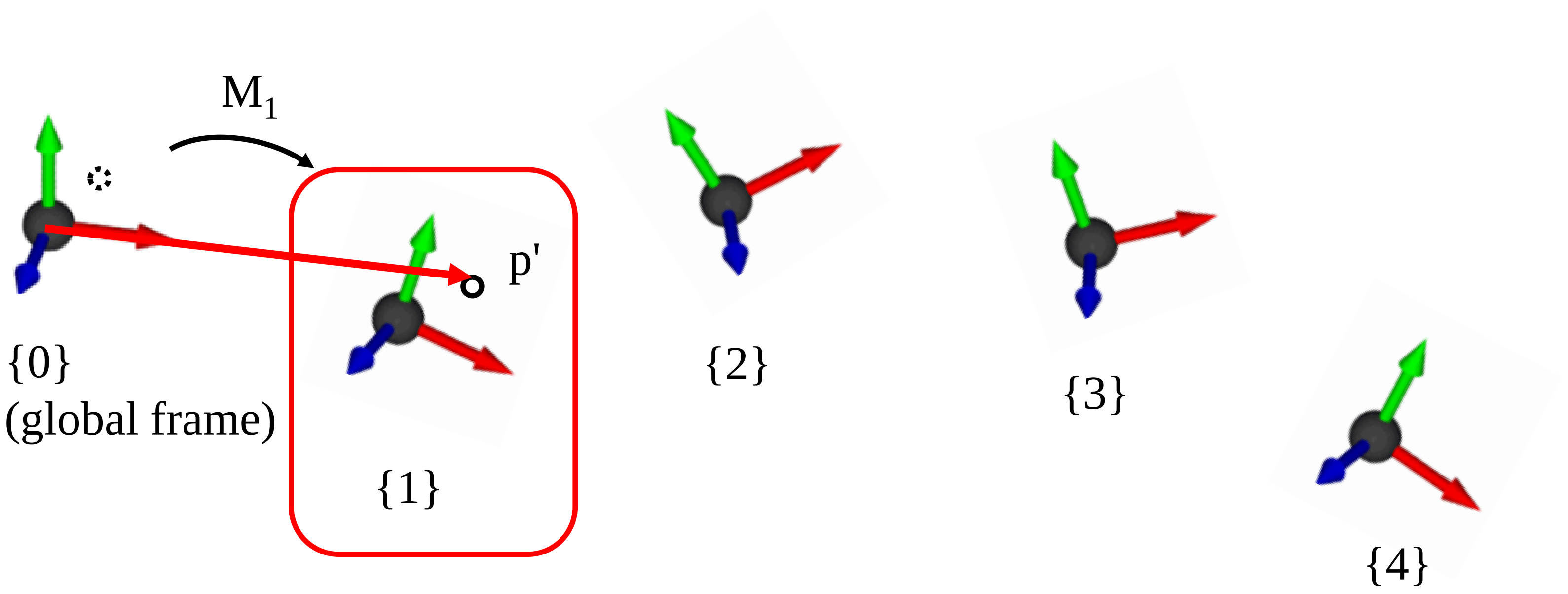
Interpretation of a Series of Transformations #2

- $p' = M_1 M_2 M_3 M_4 \, p$



Interpretation of a Series of Transformations #2

- $p' = \boxed{M_1} M_2 M_3 M_4 \, p$

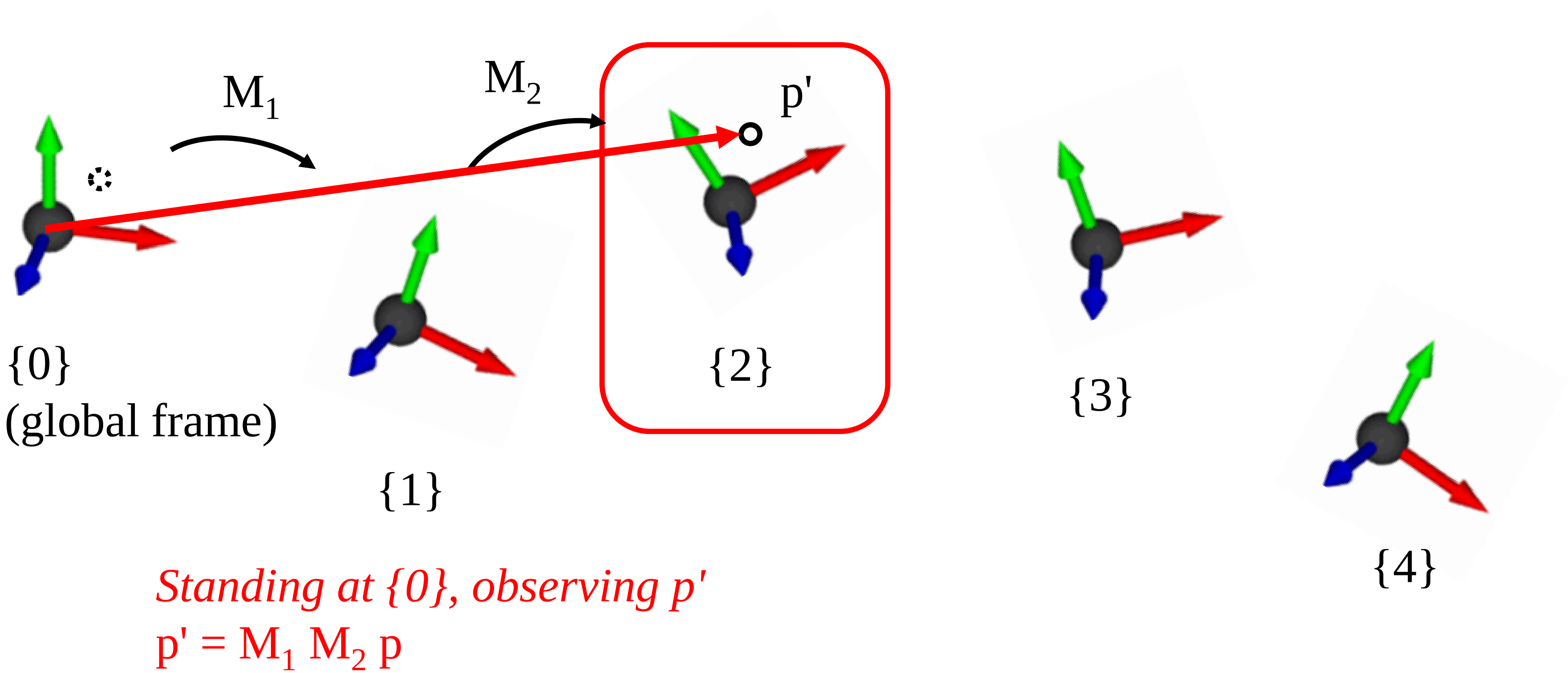


Standing at {0}, observing p'

$p' = M_1 \, p$

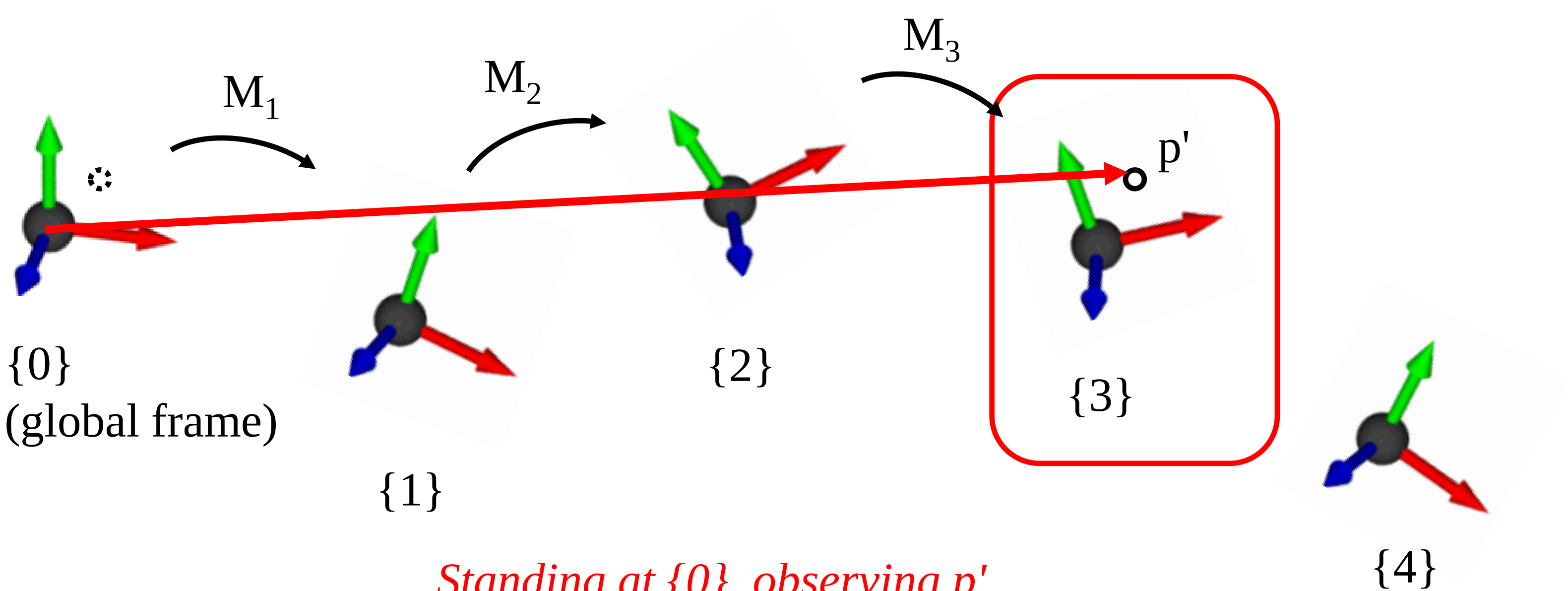
Interpretation of a Series of Transformations #2

- $p' = \boxed{M_1 M_2} M_3 M_4 \, p$



Interpretation of a Series of Transformations #2

- $p' = \boxed{M_1 M_2 M_3} M_4\ p$

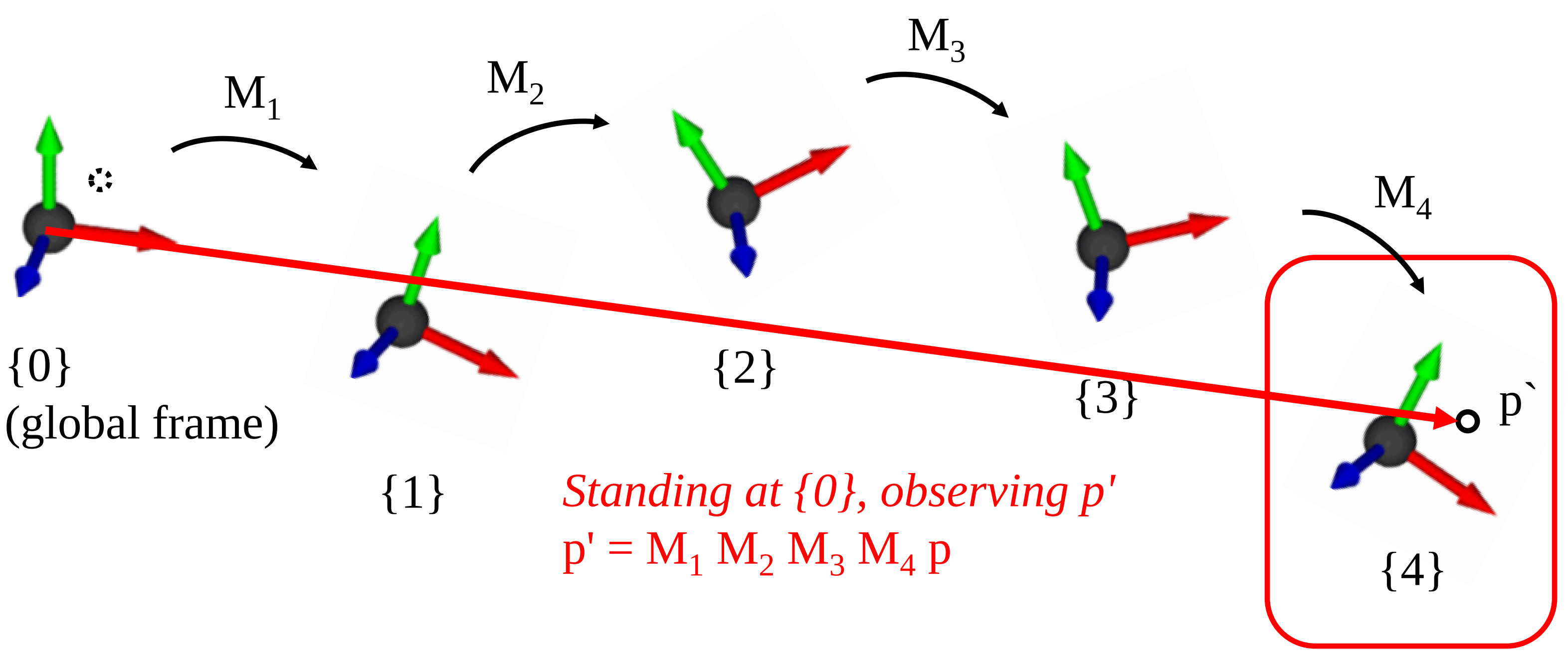


Standing at {0}, observing p'

$p' = M_1\ M_2\ M_3\ p$

Interpretation of a Series of Transformations #2

- p' = $\boxed{M_1 M_2 M_3 M_4\ p}$



Left & Right Multiplication

- Thinking it deeper, we can see:

- **p' = RTp (left-multiplication by R)**
  - (R-to-L) Apply T to a point p w.r.t. global frame.
  - **Apply R to a point Tp w.r.t. global frame.**

- **p' = TRp (right-multiplication by R)**
  - (L-to-R) Apply T to a point p w.r.t. local frame.
  - **Apply R to a point Tp w.r.t local frame.**

[Practice] Interpretation of Composite Transformations

- Just start from the Lecture 4 practice code "[Practice] OpenGL Trans. Functions".

- Differences are:

```
def drawFrame():
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([1.,0.,0.]))
    glColor3ub(0, 255, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([0.,1.,0.]))
    glColor3ub(0, 0, 255)
    glVertex3fv(np.array([0.,0.,0]))
    glVertex3fv(np.array([0.,0.,1.]))
    glEnd()
```

[Practice] Interpretation of Composite Transformations

```
def render(camAng):
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)
    glLoadIdentity()
    glOrtho(-1,1, -1,1, -1,1)
    gluLookAt(.1*np.sin(camAng),.1,.1*np.cos(camAng), 0,0,0, 0,1,0)

    # draw global frame
    drawFrame()

    # 1) p'=TRp
    glTranslatef(.4, .0, 0)
    drawFrame()     # frame defined by T
    glRotatef(60, 0, 0, 1)
    drawFrame()     # frame defined by TR

    # # 2) p'=RTp
    # glRotatef(60, 0, 0, 1)
    # drawFrame()   # frame defined by R
    # glTranslatef(.4, .0, 0)
    # drawFrame()   # frame defined by RT

    drawTriangle()
```

Quiz #2

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

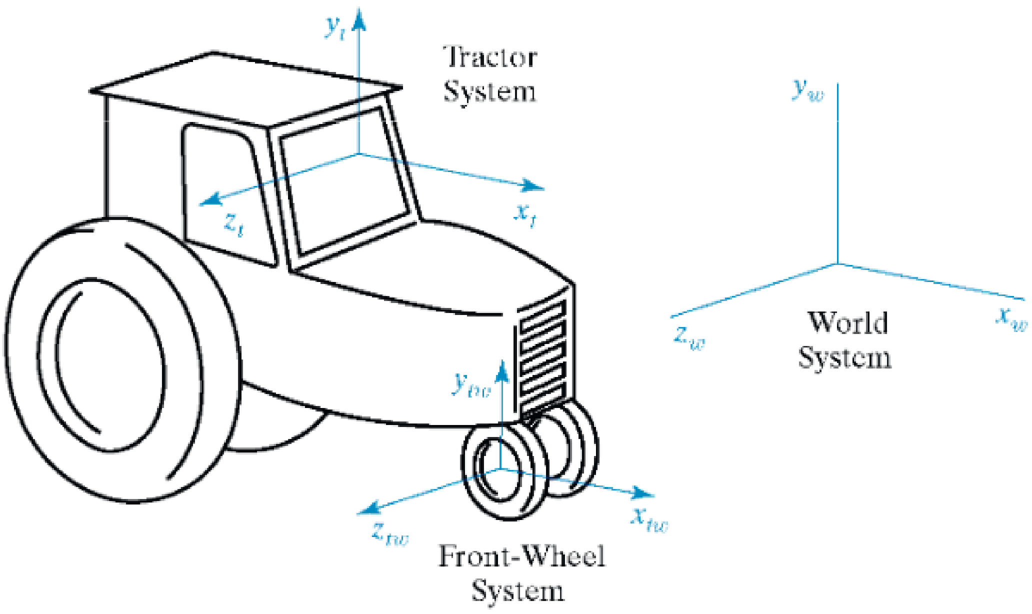
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

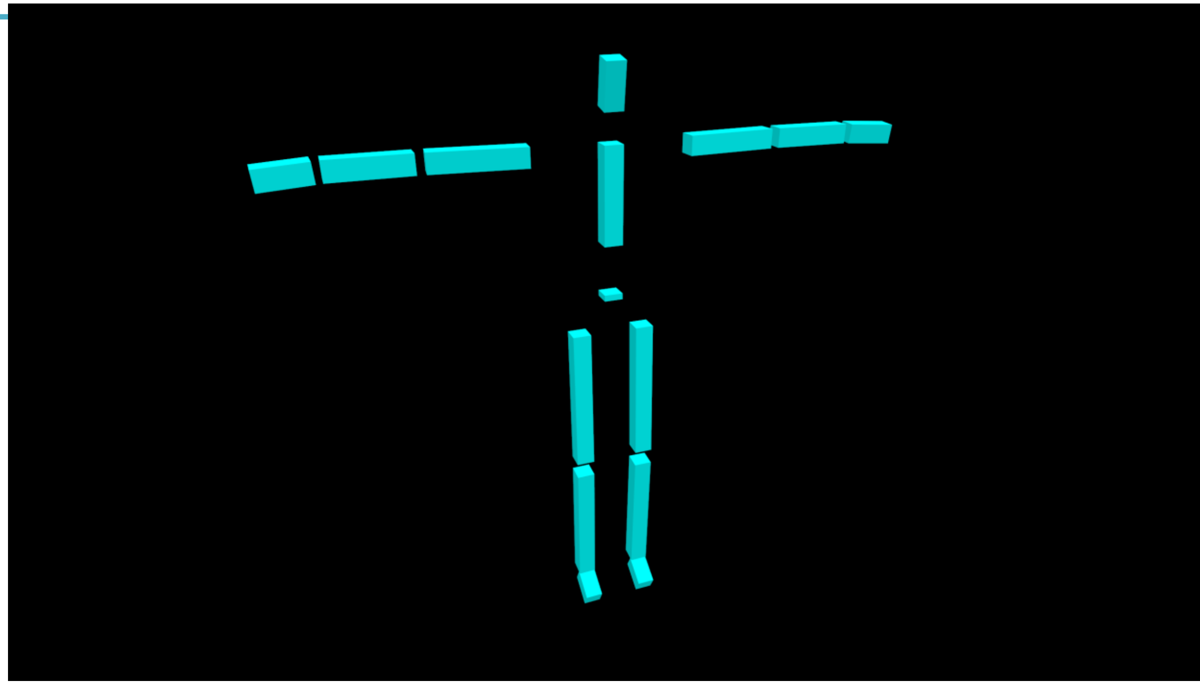
Hierarchical Modeling

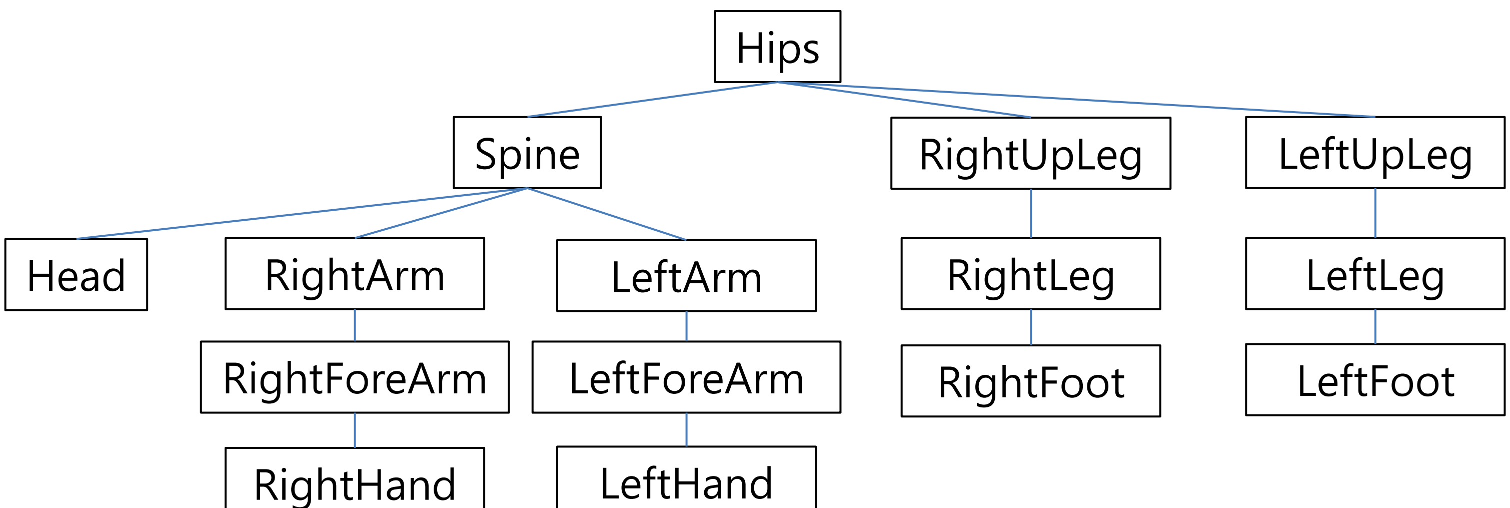
Hierarchical Modeling

- Nesting the description of subparts (child parts) into another part (parent part) to form a tree structure
- Each part has its own reference frame (local frame).
- Each part's movement is described w.r.t. its parent's reference frame.

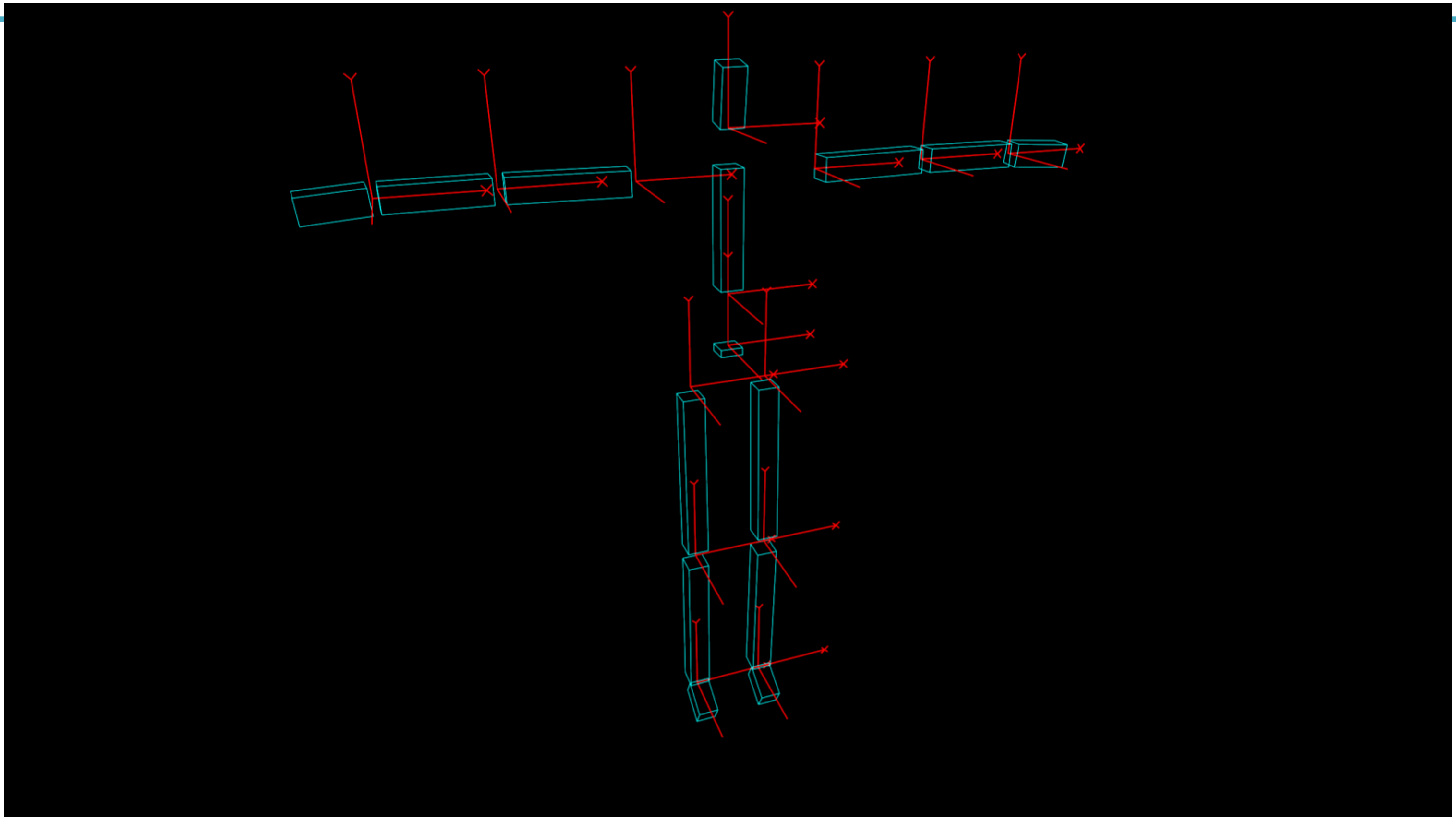


Another Example - Human Figure



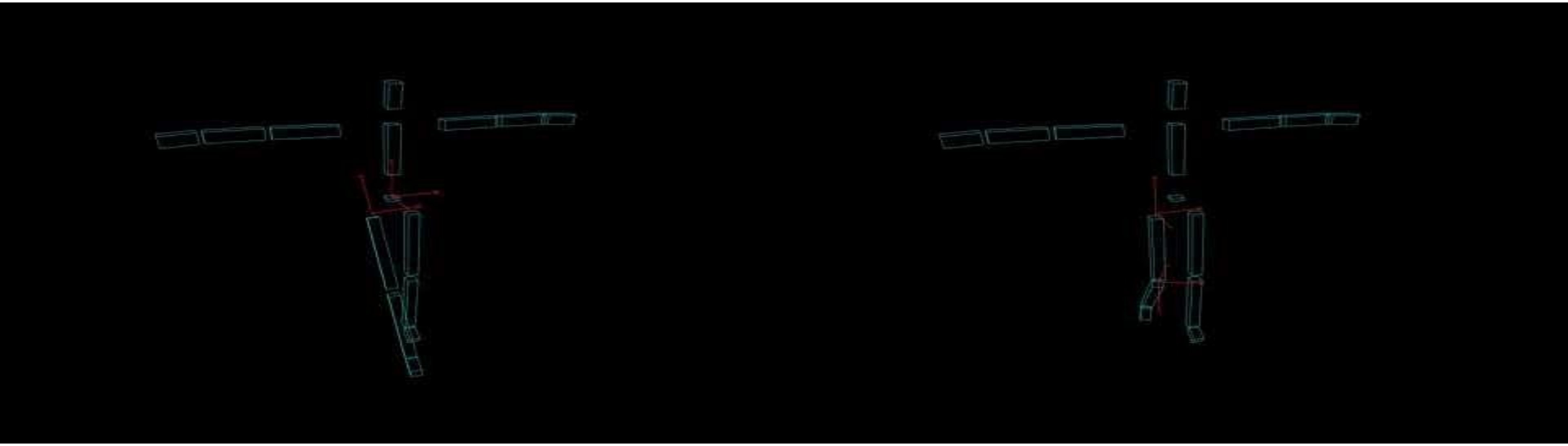


Human Figure - Frames



- Each part has its own reference frame (local frame).

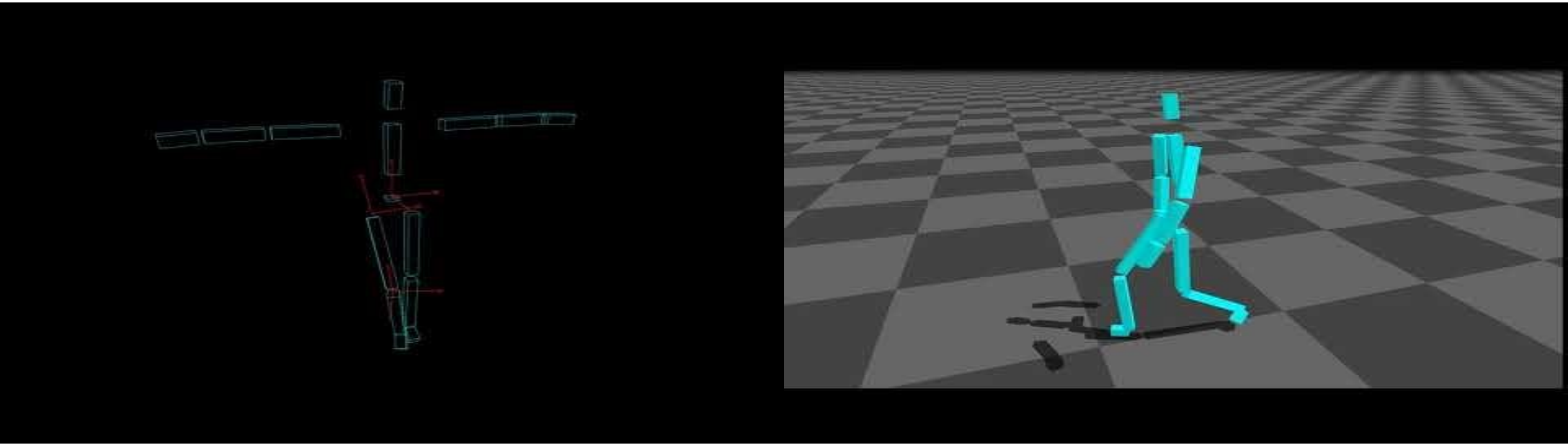
Human Figure - Movement of rhip & rknee



https://youtu.be/Q7lhvMkCSCg https://youtu.be/Q5R8WGUwpFU

- Each part's movement is described w.r.t. its parent's reference frame.
  - Each part has its own transformation w.r.t. parent part's frame
  - "Grouping"

Human Figure - Movement of more joints

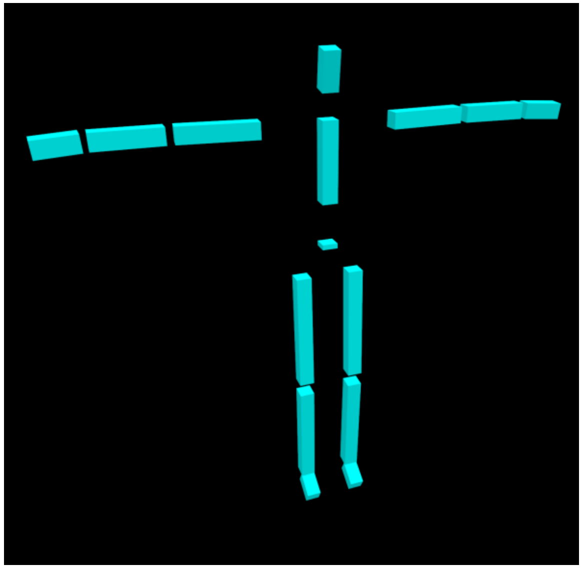


https://youtu.be/9dz8bvVK9zc https://youtu.be/PEhyWI8LGBY

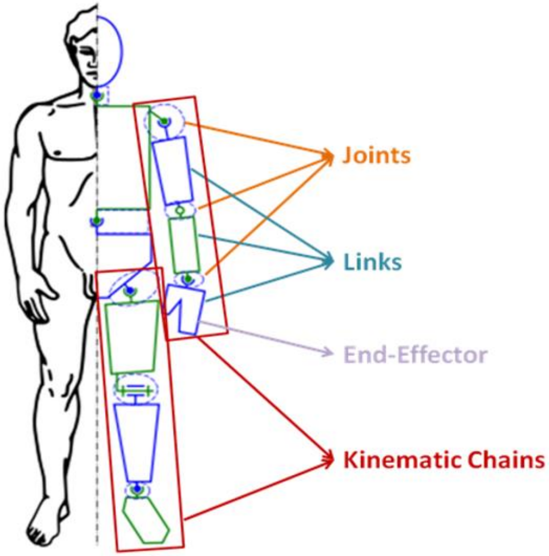
- Each part's movement is described w.r.t. its parent's reference frame.
  - Each part has its own transformation w.r.t. parent part's frame
  - "Grouping"

Articulated Body

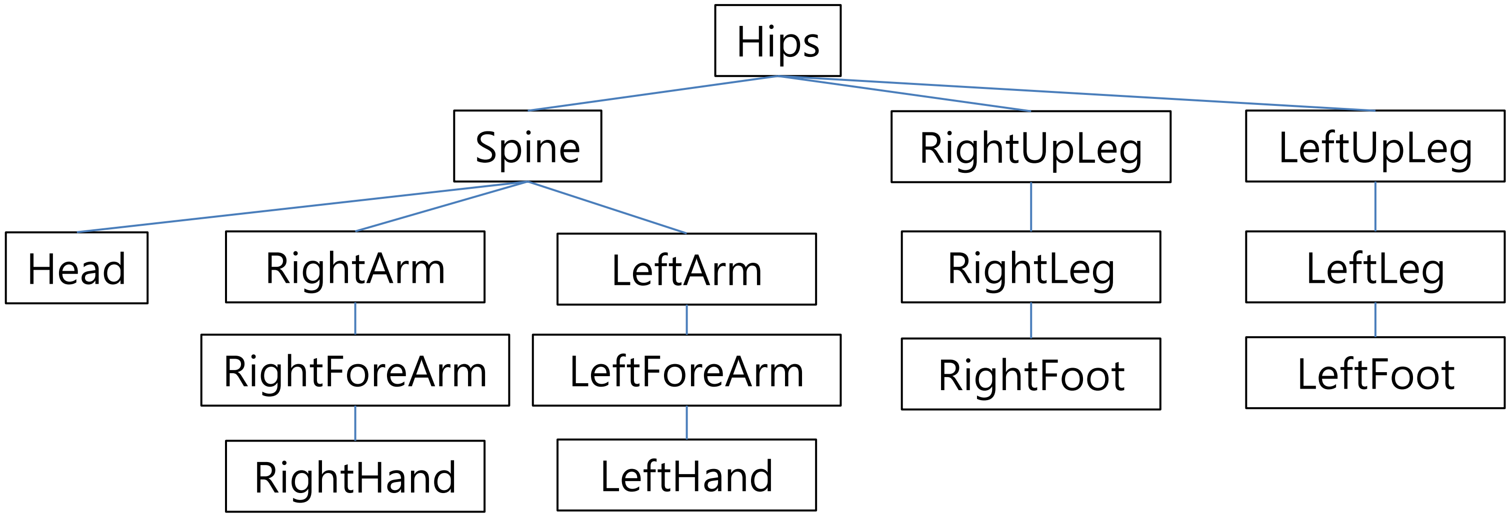
- A common type of hierarchical model used in CG is an *articulated body*
  - that has objects that are connected end to end to form multibody jointed chains.
  - a.k.a. *kinematic chain*, *linkage* (robotics)



- Terminologies
  - *Joint* - a connection between two objects which allows some motion
  - *Link* - a rigid object between joints
  - *End effector* - a free end of a kinematic chain



Articulated Body

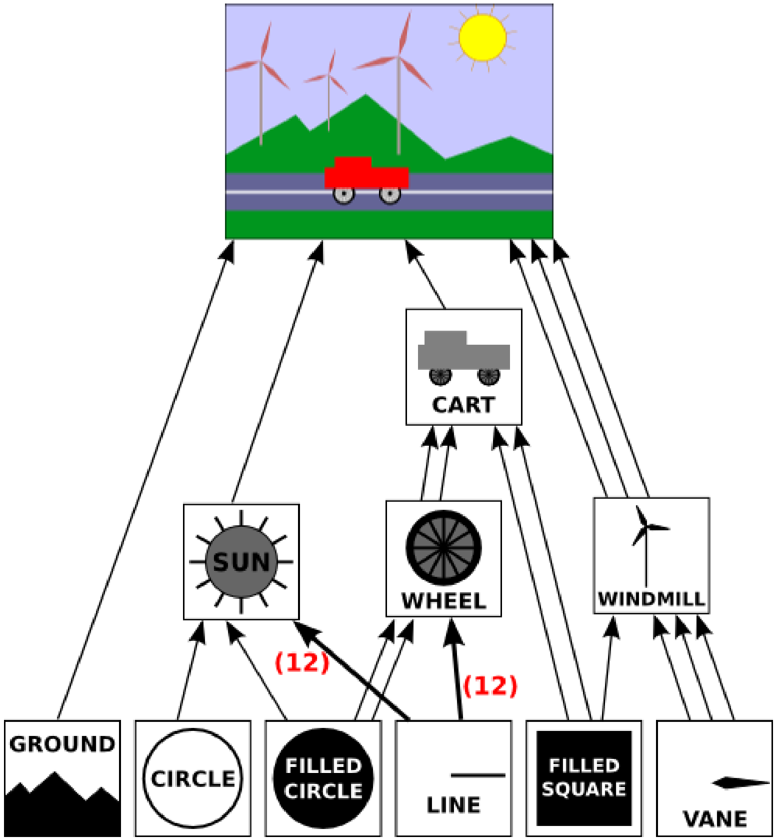


- An articulated body is represented by a graph structure.
  - A tree structure is most commonly used.

- Each node has its own transformation w.r.t. parent node’s frame

Scene Graph

- A graph structure that represents an entire scene.



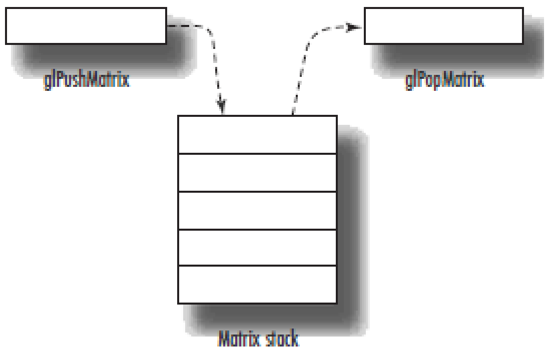
Rendering Hierarchical Models in OpenGL

- OpenGL provides a useful way of drawing objects in a hierarchical structure.

- → **Matrix stack**

OpenGL Matrix Stack

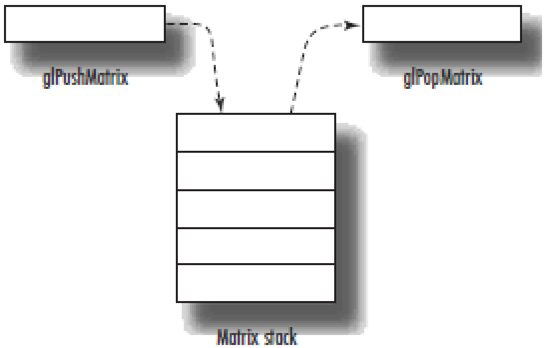
- A *stack* for transformation matrices
  - Last In First Outs



- You can **save** the **current transformation matrix** and then **restore** it after some objects have been drawn

- Useful for traversing hierarchical data structures (i.e. scene graph or tree)

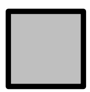
OpenGL Matrix Stack

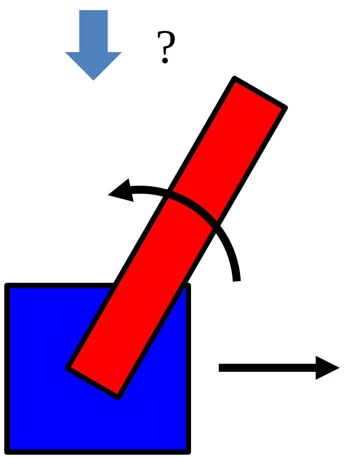


- glPushMatrix()
  - Pushes **the current matrix** onto the stack.
- glPopMatrix()
  - Pops the matrix off the stack.
- The **current matrix** is the matrix **on the top of the stack!**
- Keep in mind that the **numbers of glPushMatrix() calls and glPopMatrix() calls must be the same.**

A simple example

drawBox(): draw a unit box





Bold text is the **current transformation matrix** (the one at the top of the matrix stack)

Start with identity matrix | **I** |

glPushMatrix() | **I** | I |

glTranslate(T) # to translate base | **T** | I |

glPushMatrix() | **T** | T | I |

glScale(S) # scaling for drawing | **TS** | T | I |

drawBox() $p'=TSp$

glPopMatrix() | **T** | I |

glPushMatrix() | **T** | T | I |

glRotate(R) # to rotate arm | **TR** | T | I |

glPushMatrix() | **TR** | TR | T | I |

glScale(U) # scaling for drawing | **TRU** | TR | T | I |

drawBox() $p'=TRUp$

glPopMatrix() | **TR** | T | I |

glPopMatrix() | **T** | I |

glPopMatrix() | **I** |

[Practice] Matrix Stack

```python
import glfw
from OpenGL.GL import *
import numpy as np
from OpenGL.GLU import *

gCamAng = 0

def render(camAng):
    # enable depth test (we'll see 
details later)
    glClear(GL_COLOR_BUFFER_BIT | 
GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glLoadIdentity()

    # projection transformation
    glOrtho(-1,1, -1,1, -1,1)

    # viewing transformation
    gluLookAt(.1*np.sin(camAng),.1,
.1*np.cos(camAng), 0,0,0, 0,1,0)

    drawFrame()

    t = glfw.get_time()
```

```python
    # modeling transformation

    # blue base transformation
    glPushMatrix()
    glTranslatef(np.sin(t), 0, 0)

    # blue base drawing
    glPushMatrix()
    glScalef(.2, .2, .2)
    glColor3ub(0, 0, 255)
    drawBox()
    glPopMatrix()

    # red arm transformation
    glPushMatrix()
    glRotatef(t*(180/np.pi), 0, 0, 1)
    glTranslatef(.5, 0, .01)

    # red arm drawing
    glPushMatrix()
    glScalef(.5, .1, .1)
    glColor3ub(255, 0, 0)
    drawBox()
    glPopMatrix()

    glPopMatrix()
    glPopMatrix()
```

```
def drawBox():
    glBegin(GL_QUADS)
    glVertex3fv(np.array([1,1,0.]))
    glVertex3fv(np.array([-1,1,0.]))
    glVertex3fv(np.array([-1,-1,0.]))
    glVertex3fv(np.array([1,-1,0.]))
    glEnd()

def drawFrame():
    # draw coordinate: x in red, y in
green, z in blue
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([1.,0.,0.]))
    glColor3ub(0, 255, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([0.,1.,0.]))
    glColor3ub(0, 0, 255)
    glVertex3fv(np.array([0.,0.,0]))
    glVertex3fv(np.array([0.,0.,1.]))
    glEnd()<
```

```
def key_callback(window, key, scancode, action,
mods):
    global gCamAng, gComposedM
    if action==glfw.PRESS or
action==glfw.REPEAT:
        if key==glfw.KEY_1:
            gCamAng += np.radians(-10)
        elif key==glfw.KEY_3:
            gCamAng += np.radians(10)

def main():
    if not glfw.init():
        return
    window =
glfw.create_window(640,640,"Hierarchy",
None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)
    glfw.swap_interval(1)

    while not glfw.window_should_close(window):
        glfw.poll_events()
        render(gCamAng)
        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()
```

Quiz #3

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

OpenGL Matrix Stack Types

- Actually, OpenGL maintains four different types of matrix stacks:

- **Modelview matrix stack (GL_MODELVIEW)**
  - Stores model view matrices.
  - This is the default type (what we've just used)
- **Projection matrix stack (GL_PROJECTION)**
  - Stores projection matrices
- Texture matrix stack (GL_TEXTURE)
  - Stores transformation matrices to adjust texture coordinates. Mostly used to implement texture projection (like an image projected by a beam projector)
- Color matrix stack (GL_COLOR)
  - Rarely used. Just ignore it.

- You can switch the current matrix stack type using glMatrixMode()
  - e.g. glMatrixMode(GL_PROJECTION) to select the projection matrix stack

OpenGL Matrix Stack Types

- A common guide is something like:

```
/* Projection Transformation */
glMatrixMode(GL_PROJECTION);     /* specify the projection matrix */
glLoadIdentity();                /* initialize current value to identity */
gluPerspective(...);             /* or glOrtho(...) for orthographic */
                                 /* or glFrustrum(...), also for perspective */

/* Viewing And Modelling Transformation */
glMatrixMode(GL_MODELVIEW);      /* specify the modelview matrix */
glLoadIdentity();                /* initialize current value to identity */
gluLookAt(...);                  /* specify the viewing transformation */

glTranslate(...);                /* various modelling transformations */
glScale(...);
glRotate(...);
...
```

- **Projection transformation** functions (gluPerspective(), glOrtho(), …) should be called with **glMatrixMode(GL_PROJECTION).**

- **Modeling & viewing transformation** functions (gluLookAt(), glTranslate(), …) should be called with **glMatrixMode(GL_MODELVIEW).**

- Otherwise, you'll get wrong lighting results.

[Practice] With Correct Matrix Stack Types

```
def render(camAng):
    # enable depth test (we'll see 
details later)
    glClear(GL_COLOR_BUFFER_BIT | 
GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()

    # projection transformation
    glOrtho(-1,1, -1,1, -1,1)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    # viewing transformation
    gluLookAt(.1*np.sin(camAng),.1, 
.1*np.cos(camAng), 0,0,0, 0,1,0)

    drawFrame()
    t = glfw.get_time()
```

```
    # modeling transformation

    # blue base transformation
    glPushMatrix()
    glTranslatef(np.sin(t), 0, 0)

    # blue base drawing
    glPushMatrix()
    glScalef(.2, .2, .2)
    glColor3ub(0, 0, 255)
    drawBox()
    glPopMatrix()

    # red arm transformation
    glPushMatrix()
    glRotatef(t*(180/np.pi), 0, 0, 1)
    glTranslatef(.5, 0, .01)

    # red arm drawing
    glPushMatrix()
    glScalef(.5, .1, .1)
    glColor3ub(255, 0, 0)
    drawBox()
    glPopMatrix()

    glPopMatrix()
    glPopMatrix()
```

Next Time

- Lab in this week:
  - Lab assignment 8

- Next lecture:
  - 9 - Orientation & Rotation

- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Jehee Lee, SNU, http://mrl.snu.ac.kr/courses/CourseGraphics/index_2017spring.html
  - Prof. Taesoo Kwon, Hanyang Univ., http://calab.hanyang.ac.kr/cgi-bin/cg.cgi
  - Prof. Kayvon Fatahalian and Keenan Crane, CMU, http://15462.courses.cs.cmu.edu/fall2015/