
Computer Graphics

7 - Hierarchical Modeling, Mesh

Yoonsang Lee
Hanyang University

Spring 2025

Next Class Announcement

- Due to the reserve forces training schedule (Apr 28~30) for Hanyang University students,
- **Next week's** lecture and lab will be provided **as a recorded lecture video**. (will be uploaded to LMS)
 - If you have any questions about the lecture and lab, post them on the LMS Q&A board.
- There will be **no TA-led lab session for the assignment** next week, either.
 - If you have any questions about the assignment, post them on the LMS Q&A board.

(Updated) Midterm Exam Announcement

- Date & time: **May 7 (Wed), 6:30 - 7:30 PM**
- Place: TBA
- Scope: Lecture 2~**7** Lab 2~**7**

- **You cannot leave until 30 minutes after the start of the exam** even if you finish the exam earlier.

- That means, **you cannot enter the room after 30 minutes from the start of the exam** (do not be late, never too late!).

- Bring your **student ID card** to the exam.

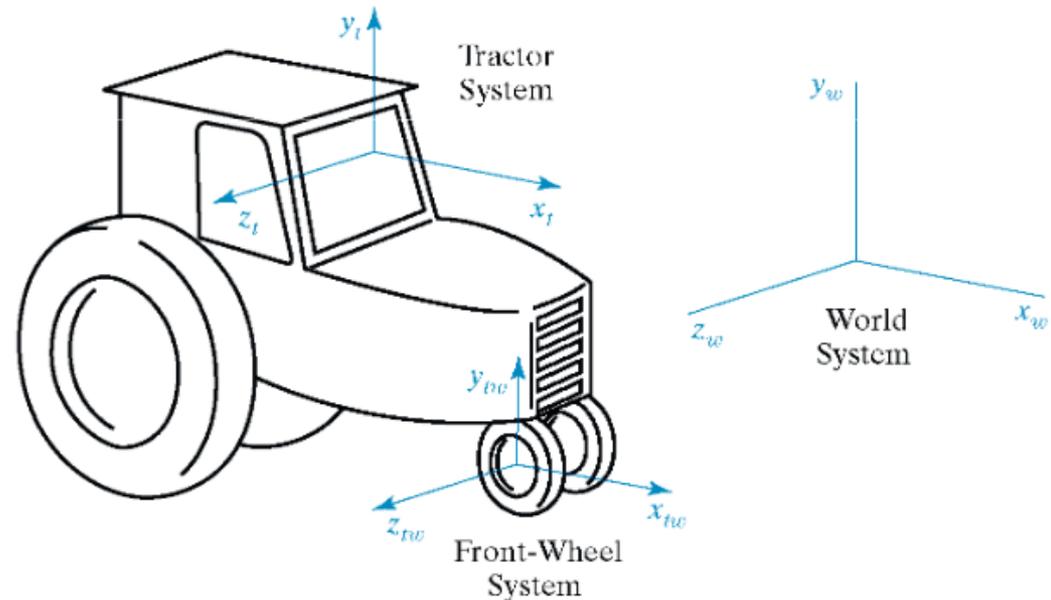
Outline

- Hierarchical Modeling
 - Concept of Hierarchical Modeling
 - Example: Human Figure
 - Rendering Hierarchical Models
 - Interpretation of a Series of Transformations
- Mesh
 - Separate triangles
 - Indexed triangle set

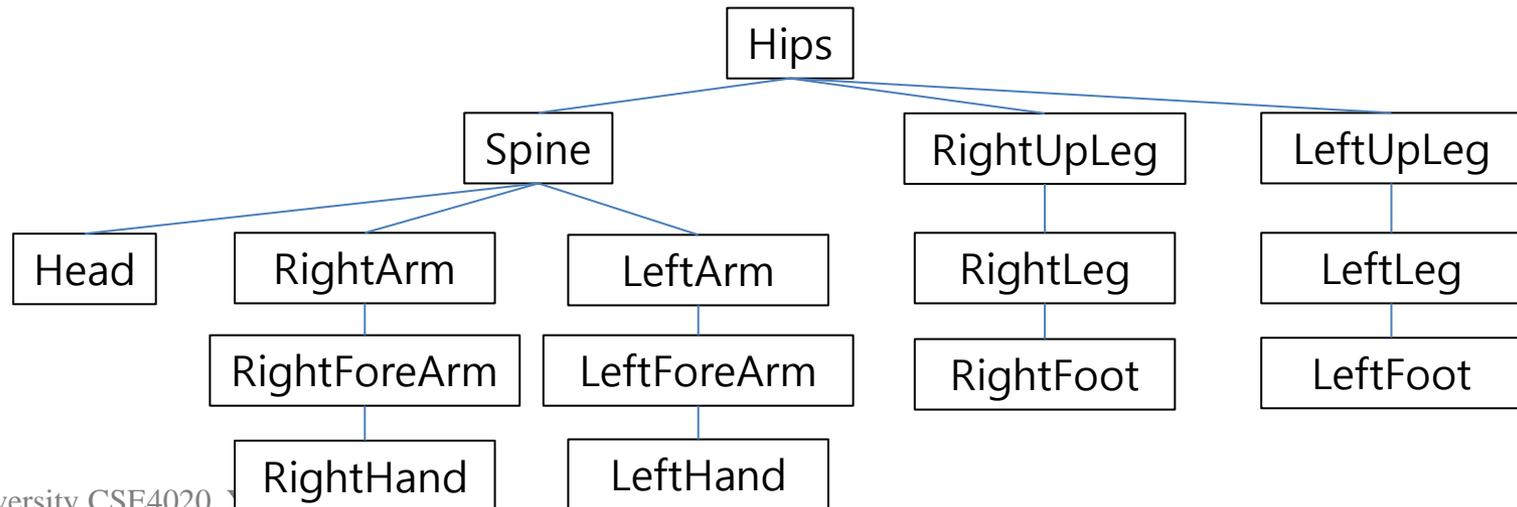
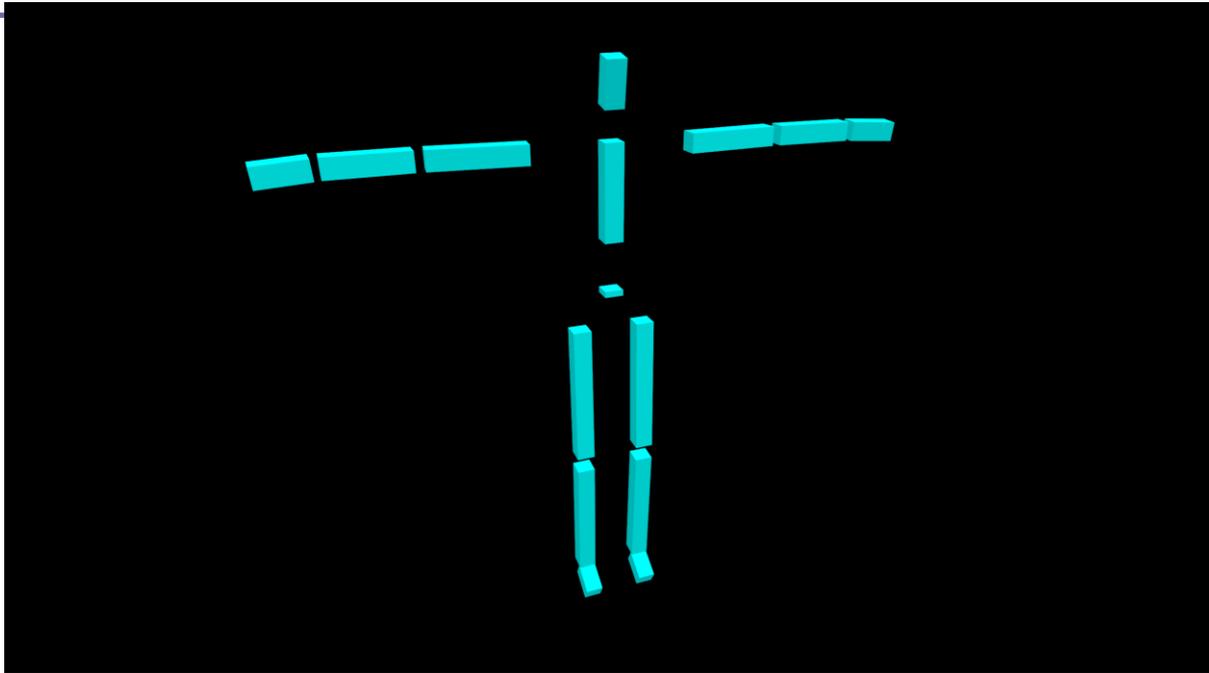
Hierarchical Modeling

Hierarchical Modeling

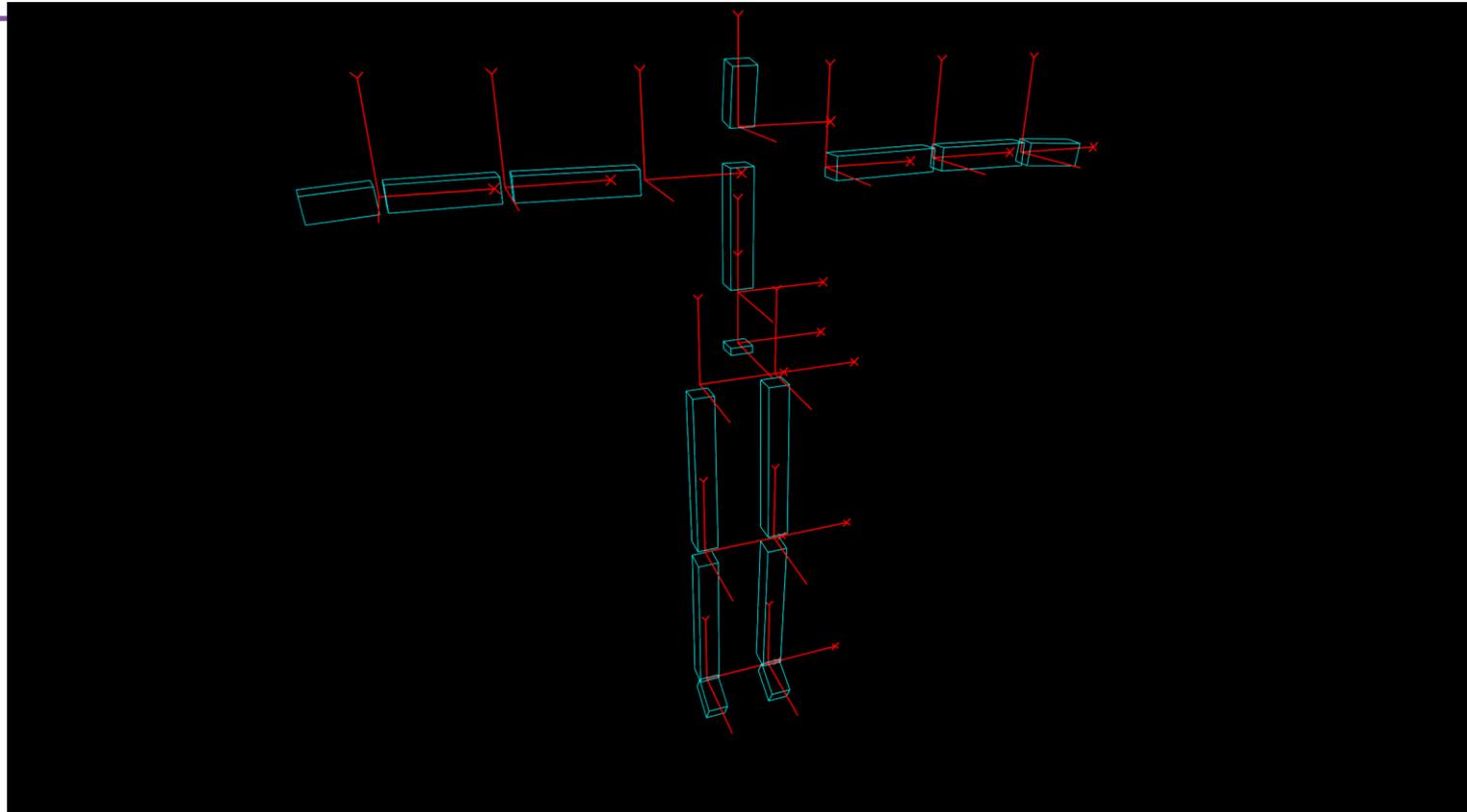
- Nesting the description of subparts (child parts) into another part (parent part) to form a tree structure.
- Each part has its own body frame.
- Each part's movement is described w.r.t. its parent's body frame.



Example - Human Figure

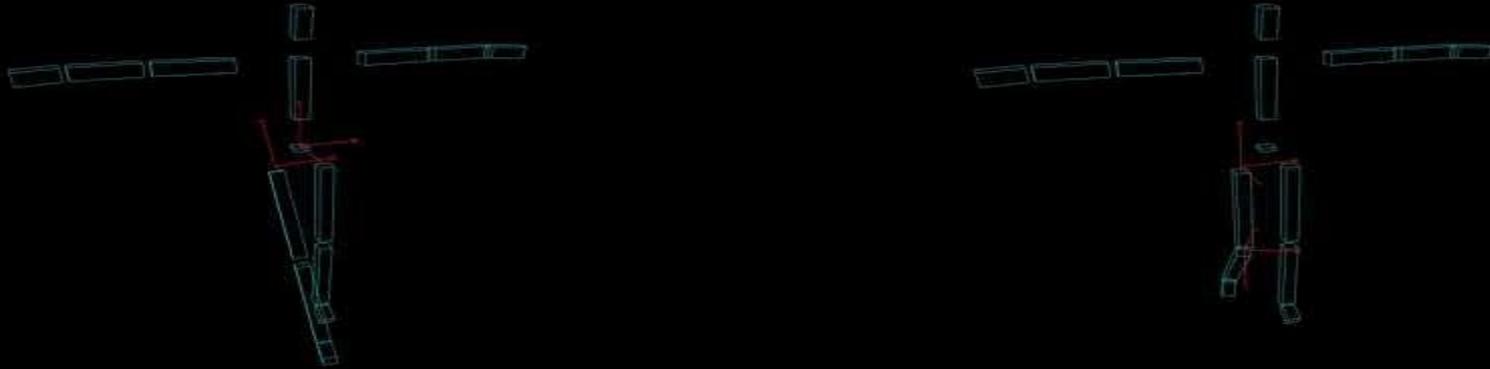


Human Figure - Frames



- Each part has its own body frame.

Human Figure - Movement of rhip & rknee

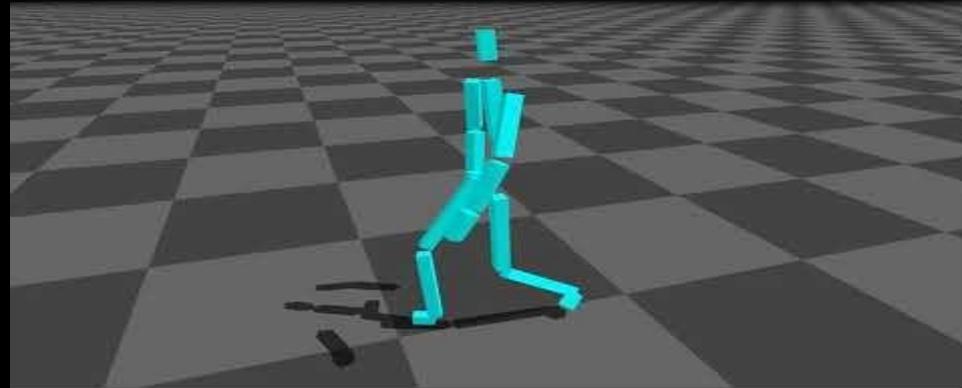
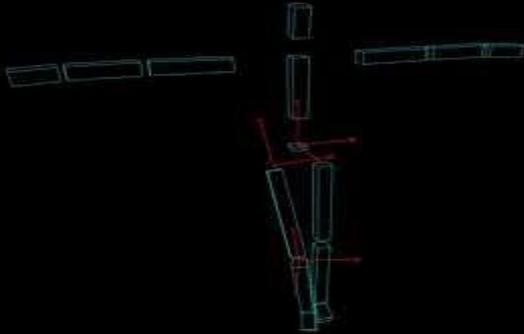


<https://youtu.be/Q7lhvMkCSCg>

<https://youtu.be/Q5R8WGUwpFU>

- Each part's movement is described w.r.t. its parent's frame.
- → Each part has its **own transformation** w.r.t. parent's frame.
- This allows a part to "group" its children together.

Human Figure - Movement of more joints

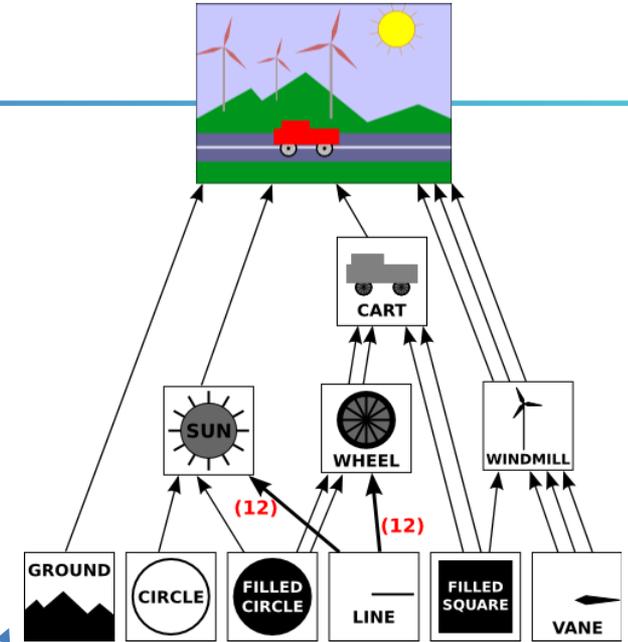
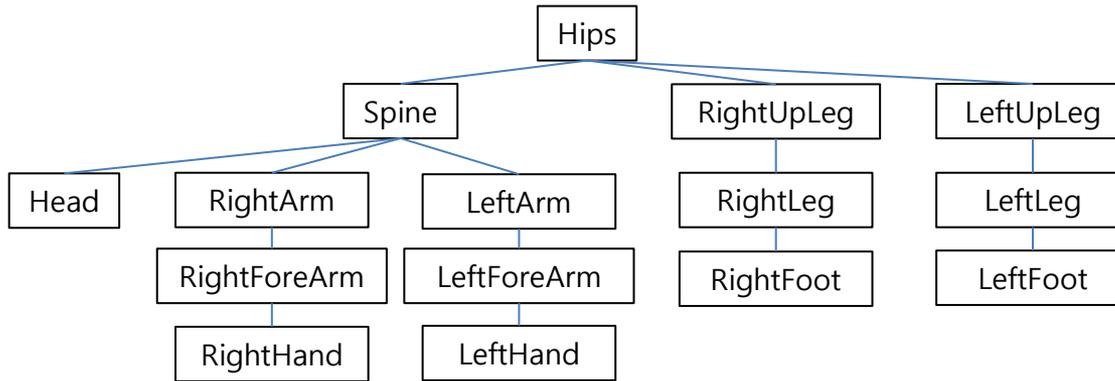


<https://youtu.be/9dz8bvVK9zc>

<https://youtu.be/PEhyWI8LGBY>

- Each part's movement is described w.r.t. its parent's frame.
- → Each part has its **own transformation** w.r.t. parent's frame.
- This allows a part to "group" its children together.

Hierarchical Model

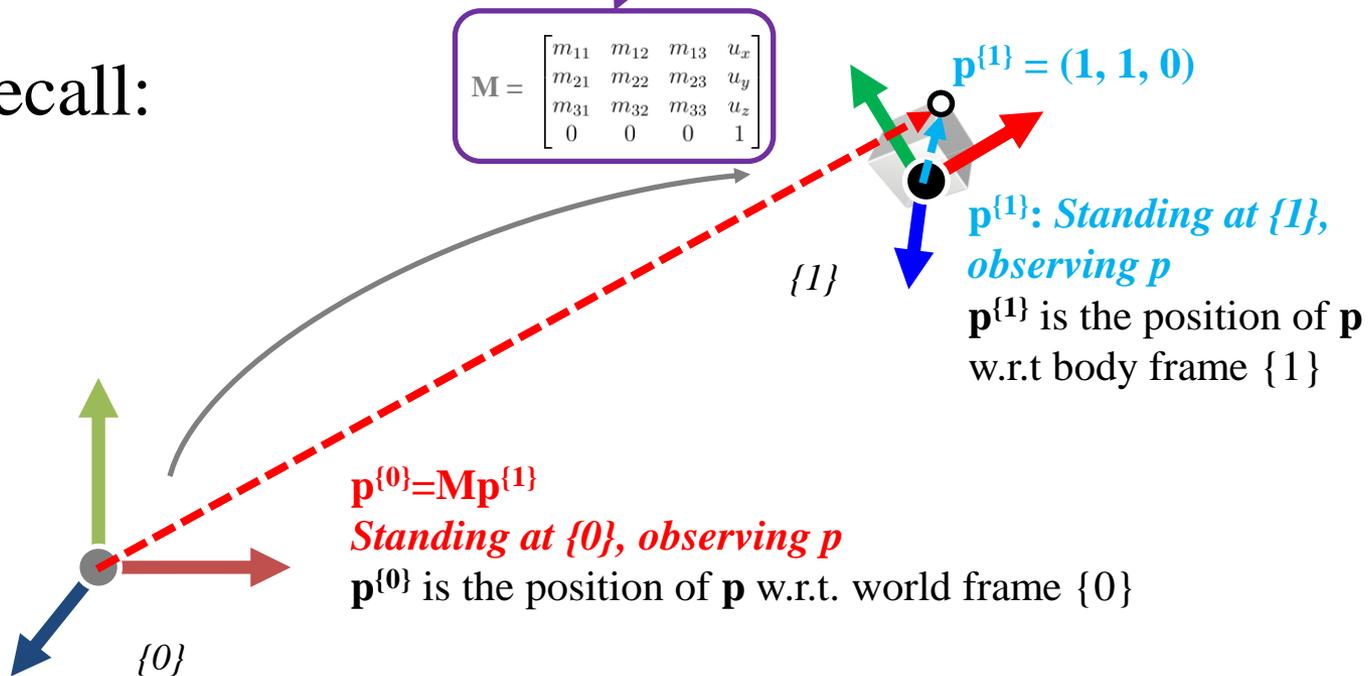


- A hierarchical model is usually represented by a **tree structure**.
- Another example of hierarchical model is *scene graph*, a graph structure that represents an entire scene.
- Each node has its **own transformation** w.r.t. parent node's frame.

Rendering Hierarchical Models

- To render a hierarchical model, we need each node's frame represented w.r.t. world frame, to compute the global position of each vertex.

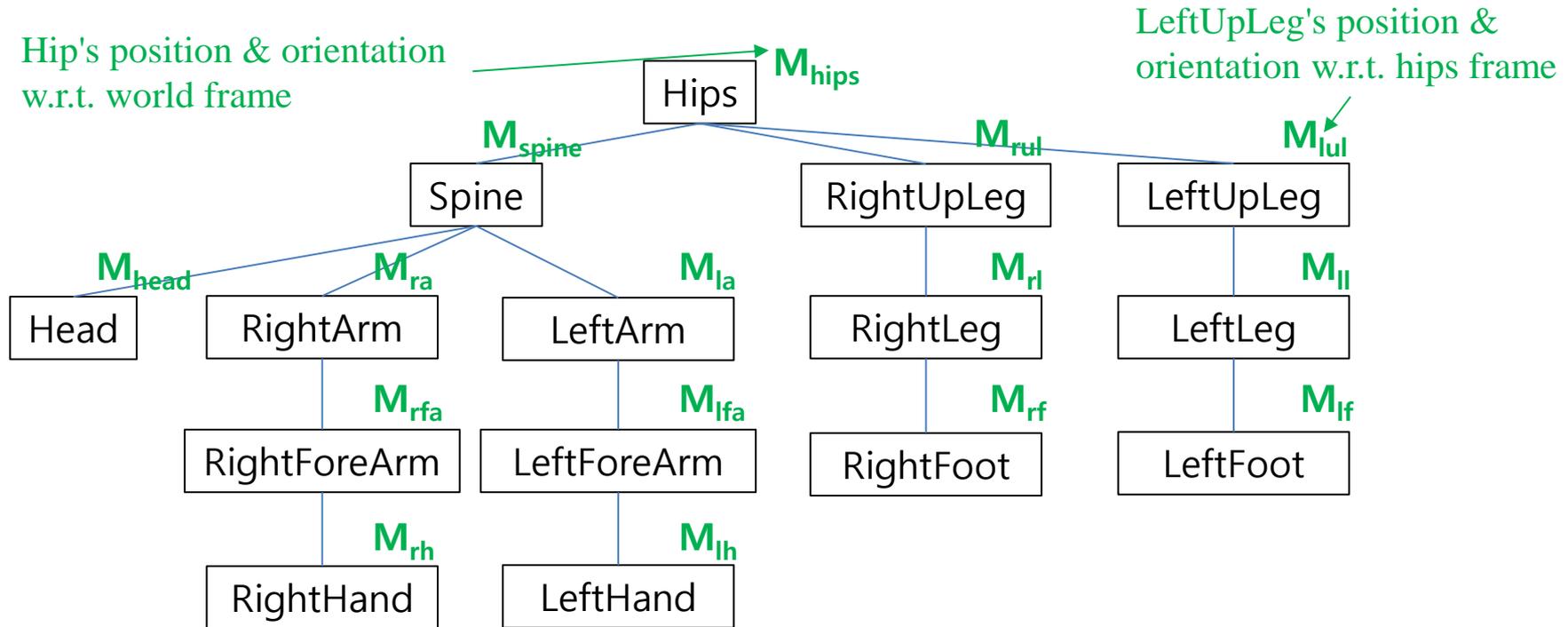
- Recall:



Rendering Hierarchical Models

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

→ **Local transformation**

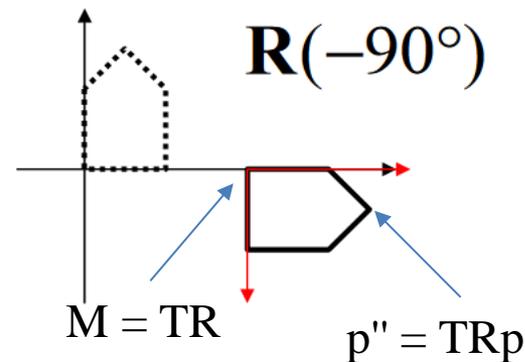
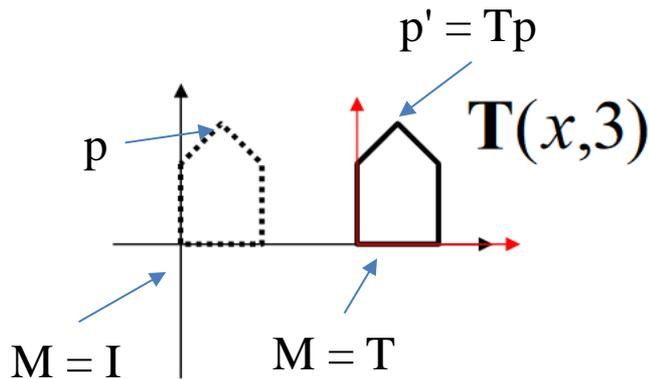


Rendering Hierarchical Models

- We need each node's frame represented w.r.t. world frame to render a hierarchical model.
→ **Global transformation**
- How can we compute the **global transform** of a node using the **local transforms** of other nodes?

Recall: Post-(right) Multiplication

- $\mathbf{p}' = \mathbf{M}_1\mathbf{M}_2\mathbf{p}$ (**post-multiplication by \mathbf{M}_2**)
 - (L-to-R)
 - 1') Apply \mathbf{M}_1 w.r.t. **body frame I** (world frame) to **update body frame to \mathbf{M}_1**
 - 2') Apply \mathbf{M}_2 w.r.t. **body frame \mathbf{M}_1** to **update body frame to $\mathbf{M}_1\mathbf{M}_2$**
 - 3') Locate \mathbf{p} in body frame $\mathbf{M}_1\mathbf{M}_2$

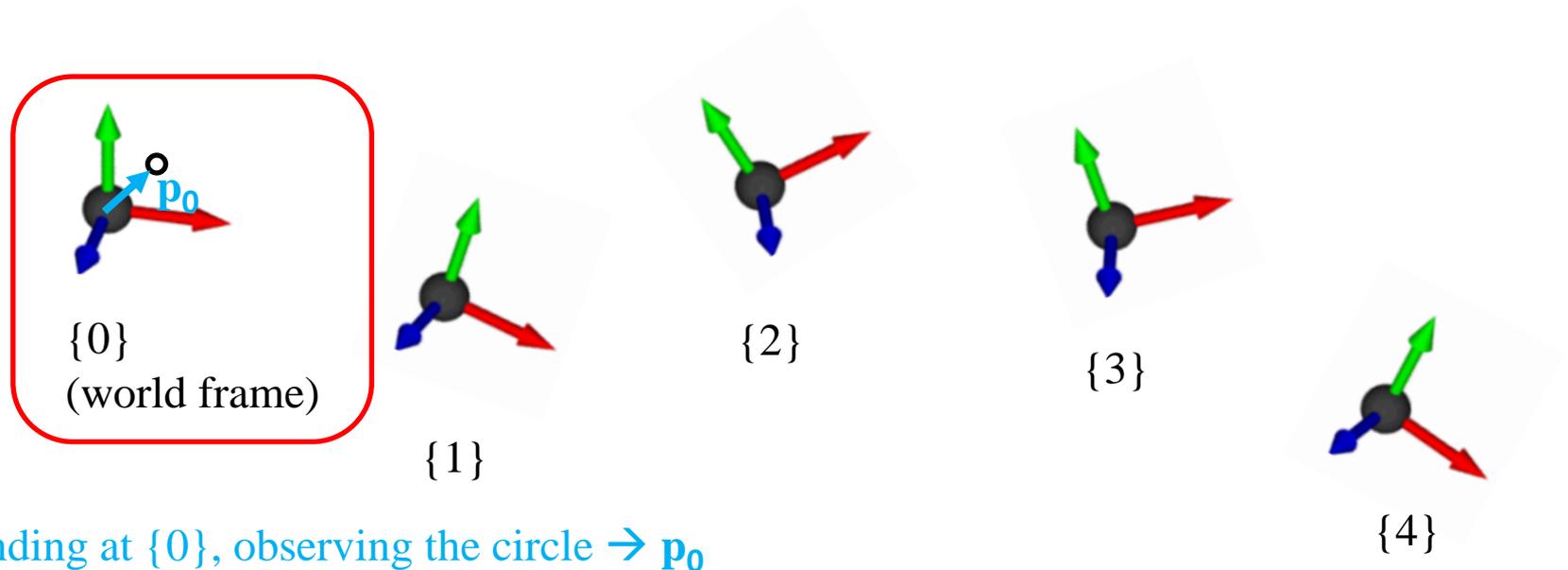


(M defines the **body frame** w.r.t. world frame)

Interpretation of a Series of Transformations

- $\mathbf{p}_0 = \mathbf{I} \mathbf{p}_0$

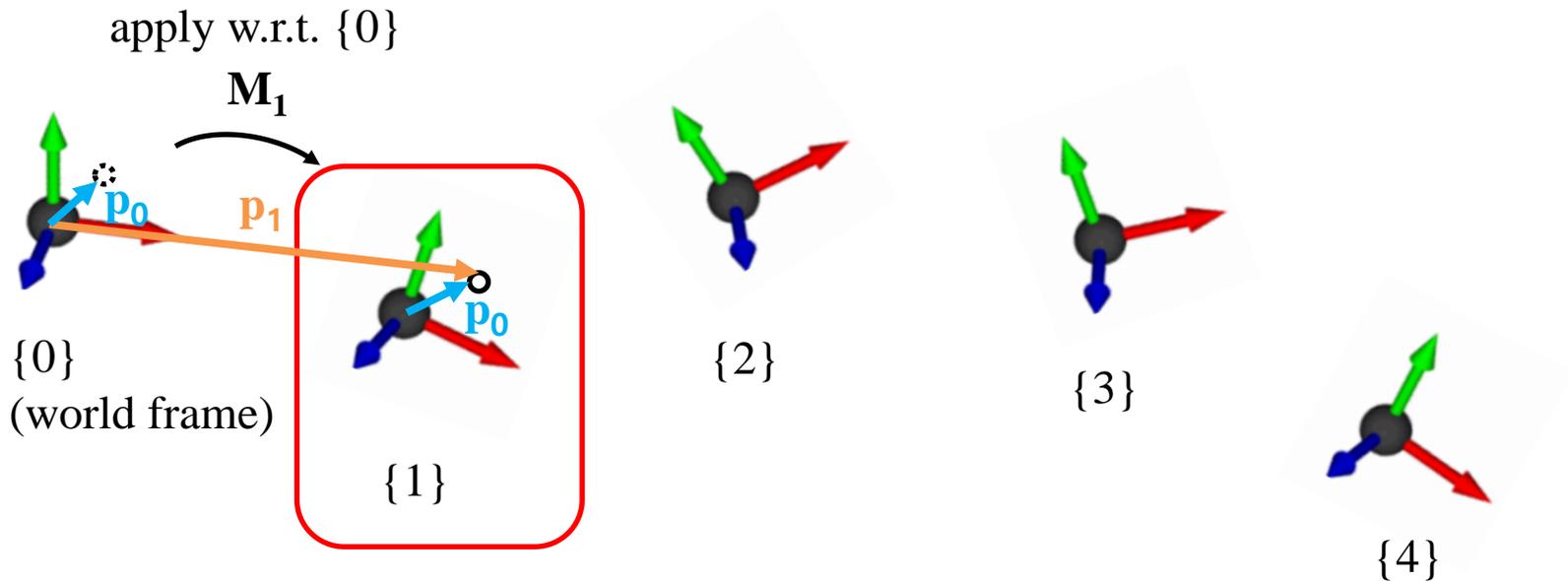
Current body frame $\{0\}$ defined w.r.t. $\{0\}$



Interpretation of a Series of Transformations

- $\mathbf{p}_1 = \underline{\mathbf{M}}_1 \mathbf{p}_0$

Current body frame {1} defined w.r.t. {0}



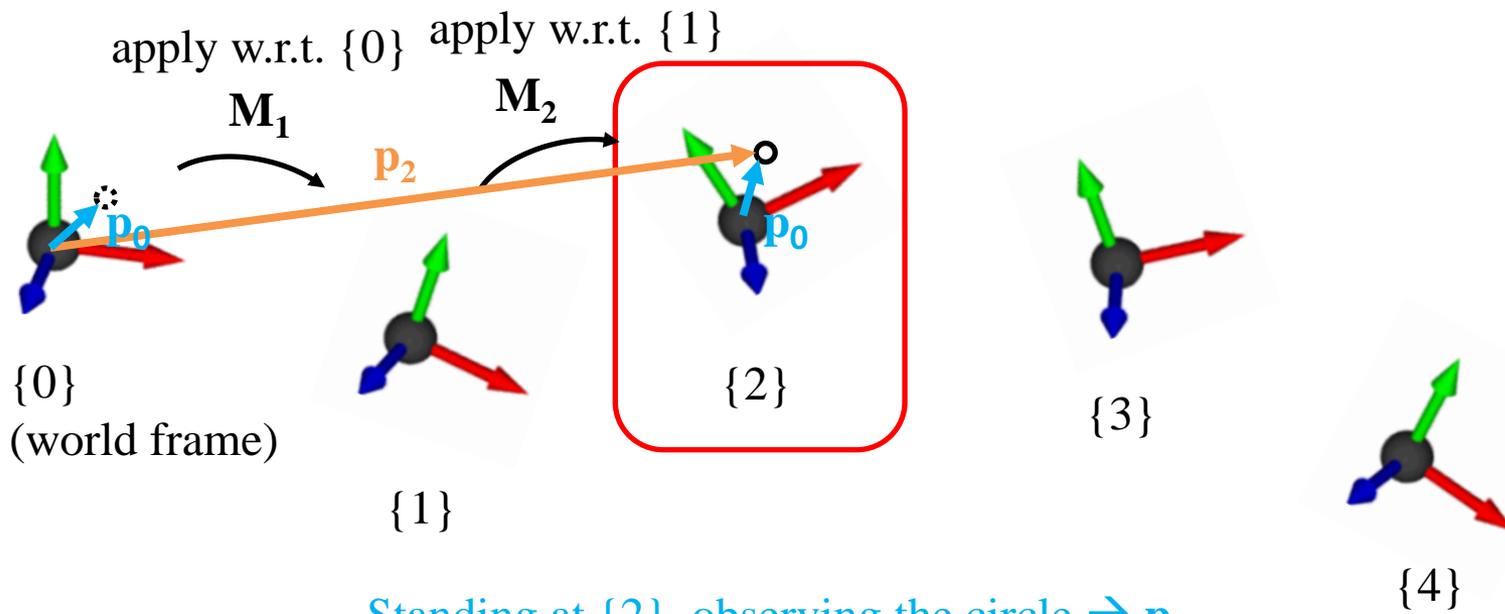
Standing at {1}, observing the circle $\rightarrow p_0$

Standing at {0}, observing the circle $\rightarrow p_1$

Interpretation of a Series of Transformations

- $\mathbf{p}_2 = \underline{\mathbf{M}_1 \mathbf{M}_2} \mathbf{p}_0$

Current body frame {2} defined w.r.t. {0}

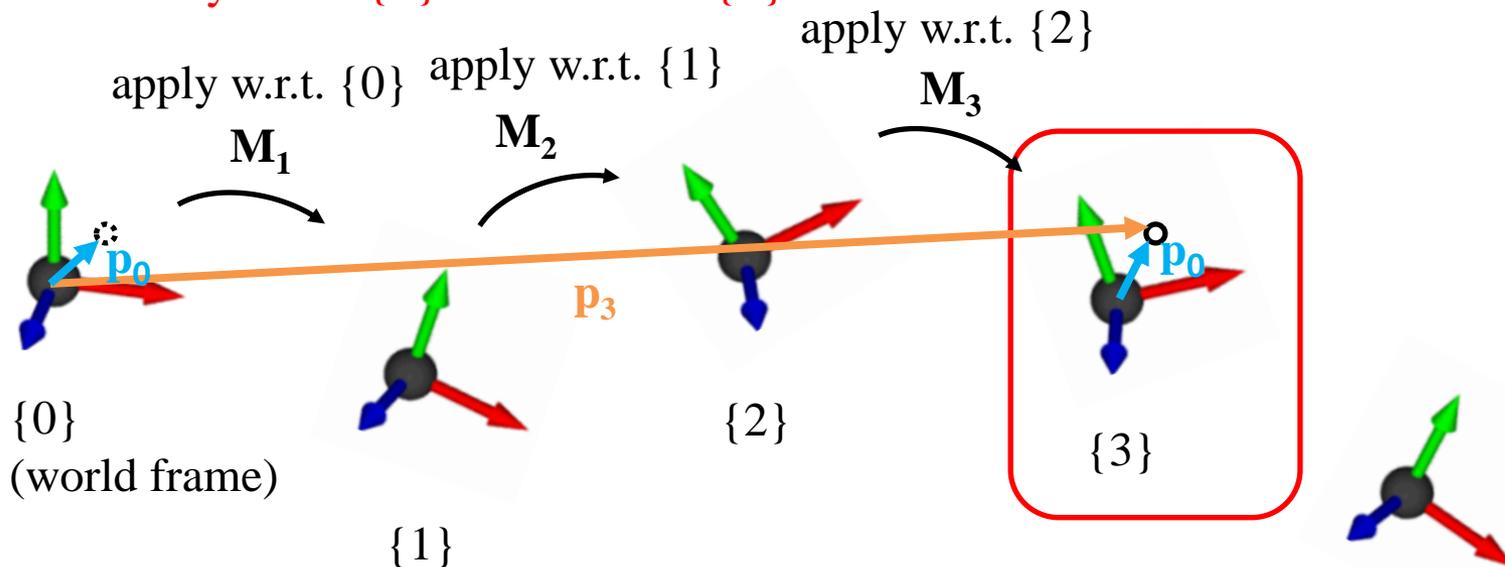


Standing at {2}, observing the circle $\rightarrow \mathbf{p}_0$
Standing at {0}, observing the circle $\rightarrow \mathbf{p}_2$

Interpretation of a Series of Transformations

- $\mathbf{p}_3 = \underline{\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3} \mathbf{p}_0$

Current body frame {3} defined w.r.t. {0}

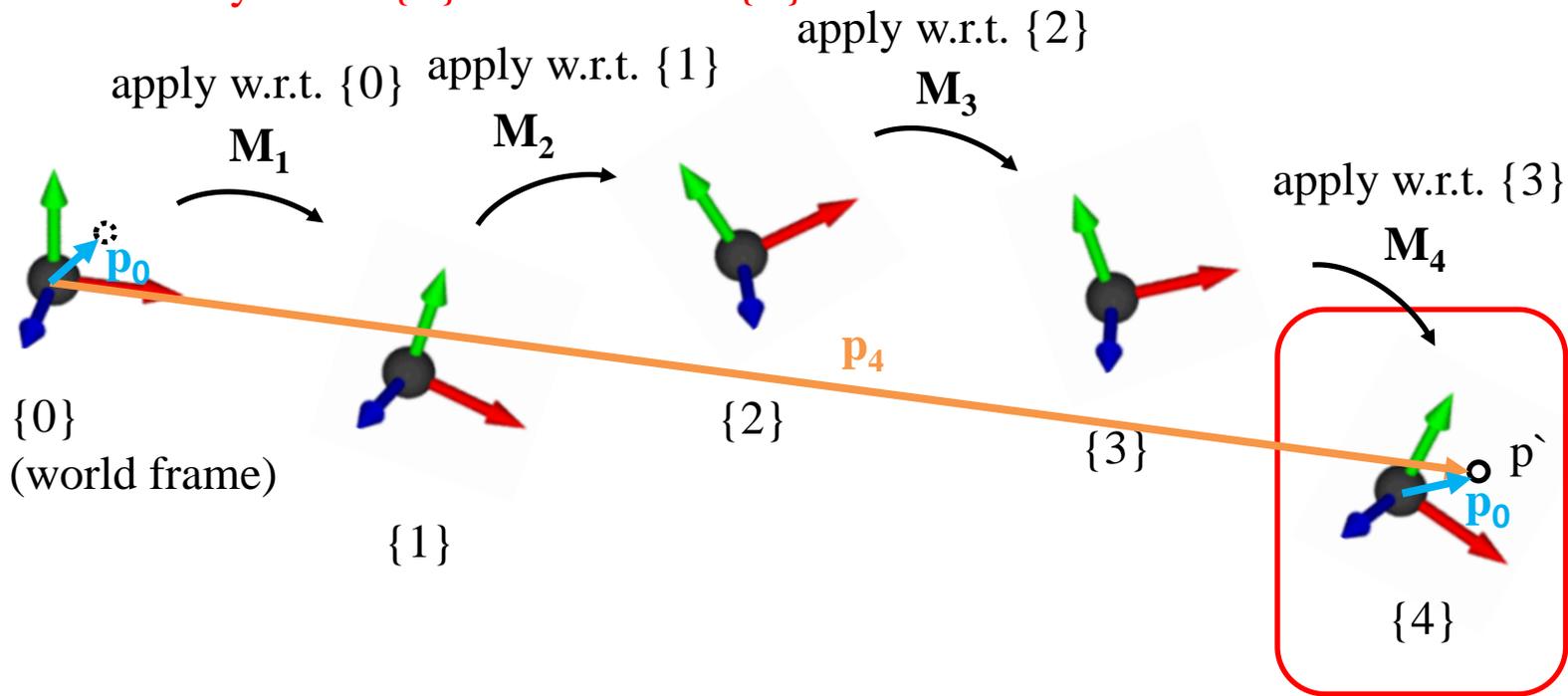


Standing at {3}, observing the circle $\rightarrow p_0$
Standing at {0}, observing the circle $\rightarrow p_3$

Interpretation of a Series of Transformations

- $\mathbf{p}_4 = \underline{\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\mathbf{M}_4} \mathbf{p}_0$

Current body frame {4} defined w.r.t. {0}



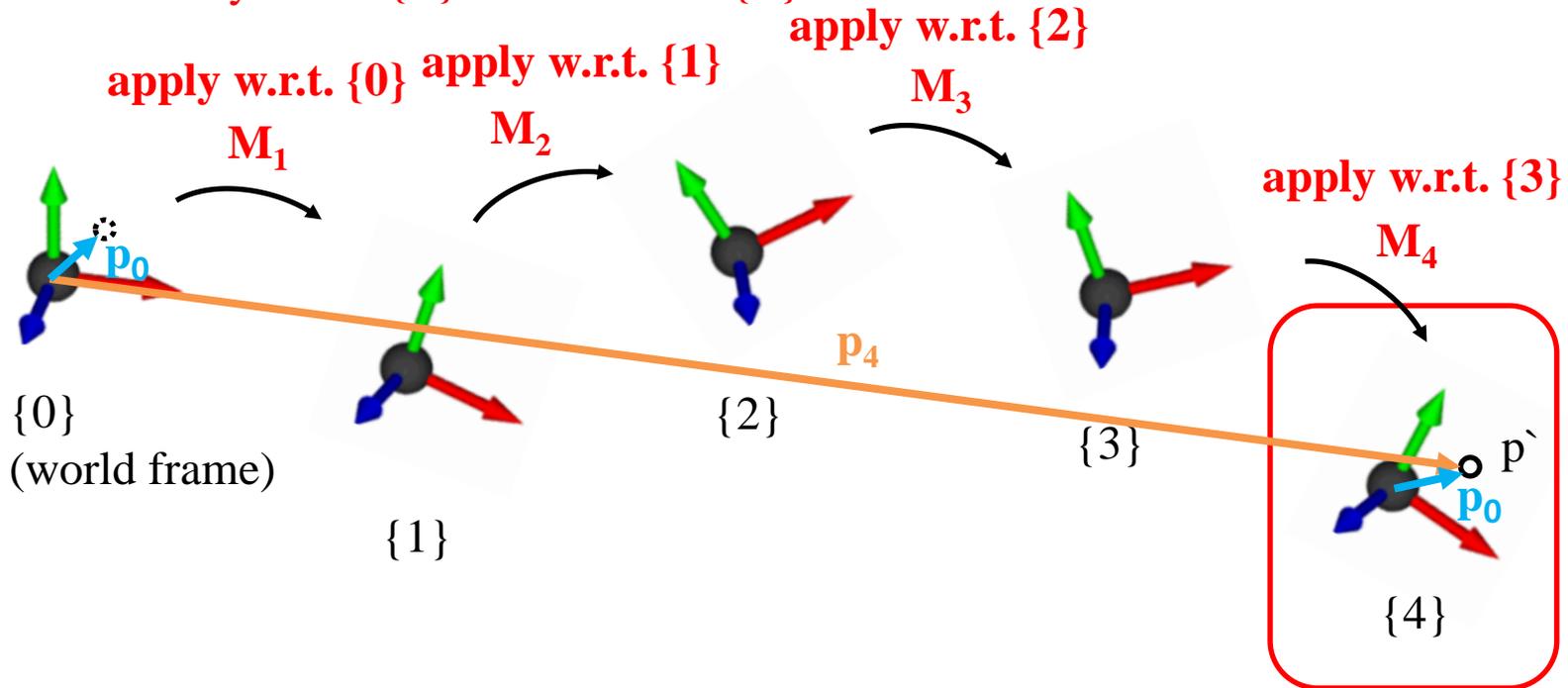
Standing at {4}, observing the circle $\rightarrow p_0$

Standing at {0}, observing the circle $\rightarrow p_4$

Interpretation of a Series of Transformations

- $\mathbf{p}_4 = \underline{\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\mathbf{M}_4} \mathbf{p}_0$

Current body frame {4} defined w.r.t. {0}



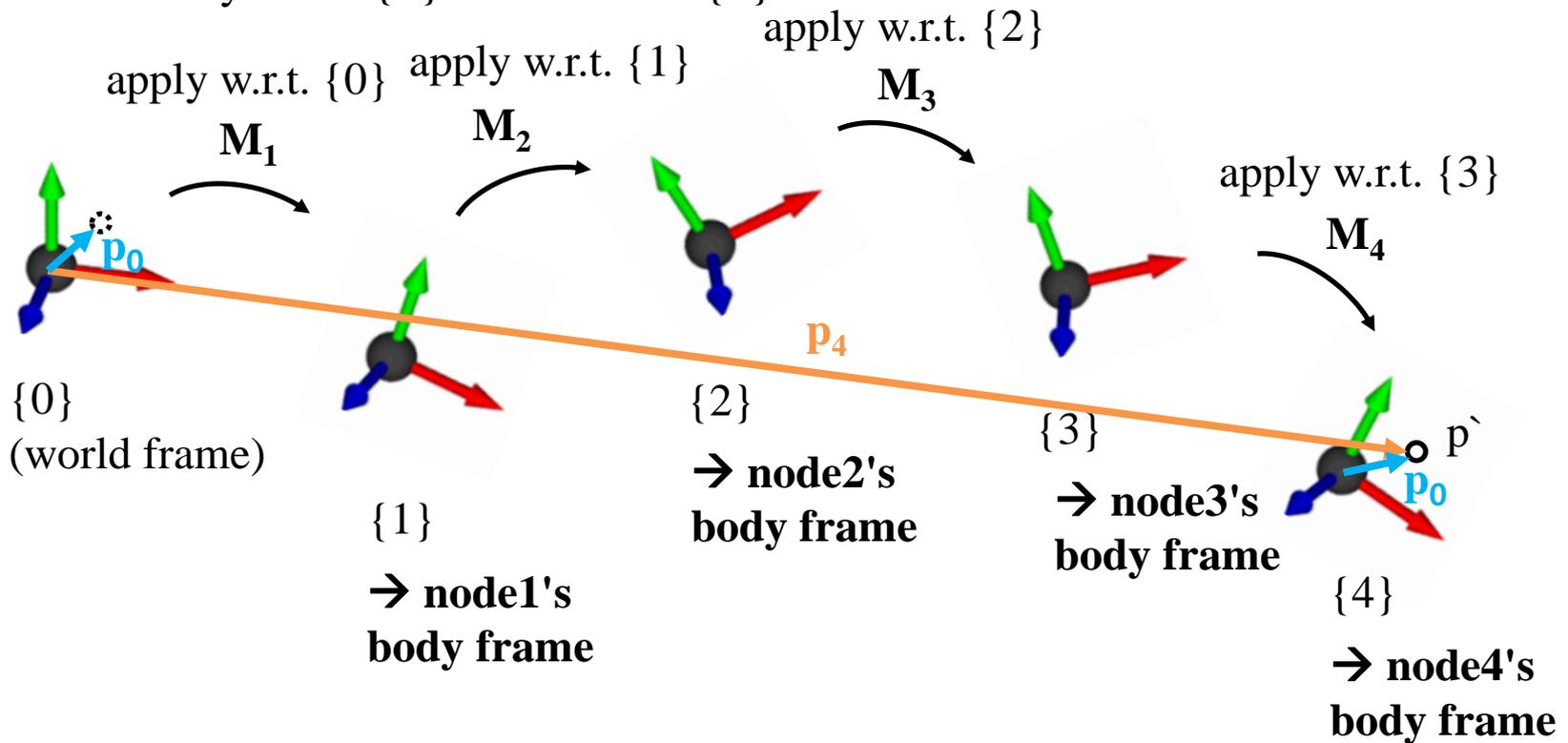
Standing at $\{4\}$, observing the circle $\rightarrow p_0$

Standing at $\{0\}$, observing the circle $\rightarrow p_4$

Interpretation of a Series of Transformations

- $\mathbf{p}_4 = \underline{\mathbf{M}_1\mathbf{M}_2\mathbf{M}_3\mathbf{M}_4} \mathbf{p}_0$

Current body frame {4} defined w.r.t. {0}

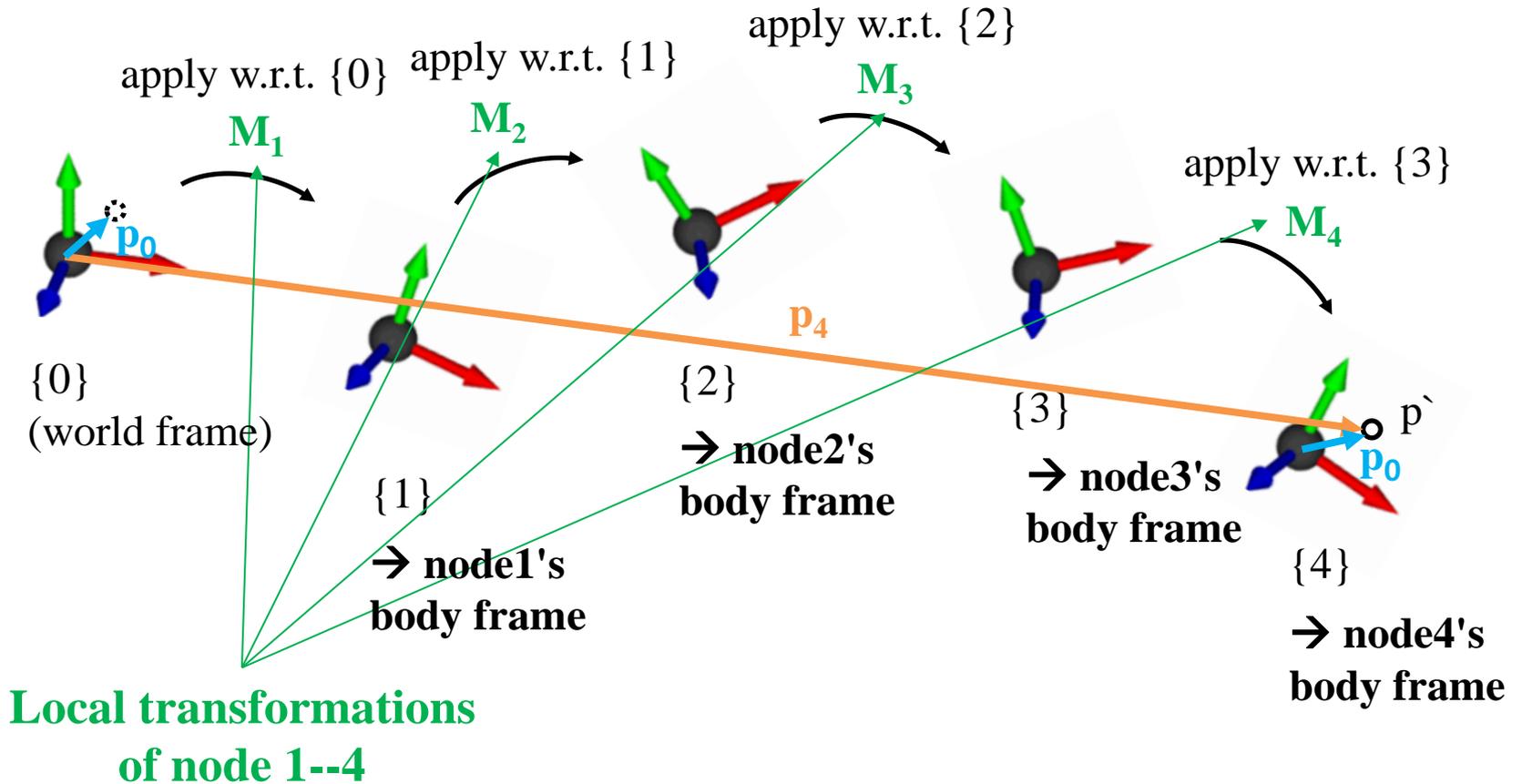


Computing Global Transform from Series of Local Transforms

Global transformation of node 4

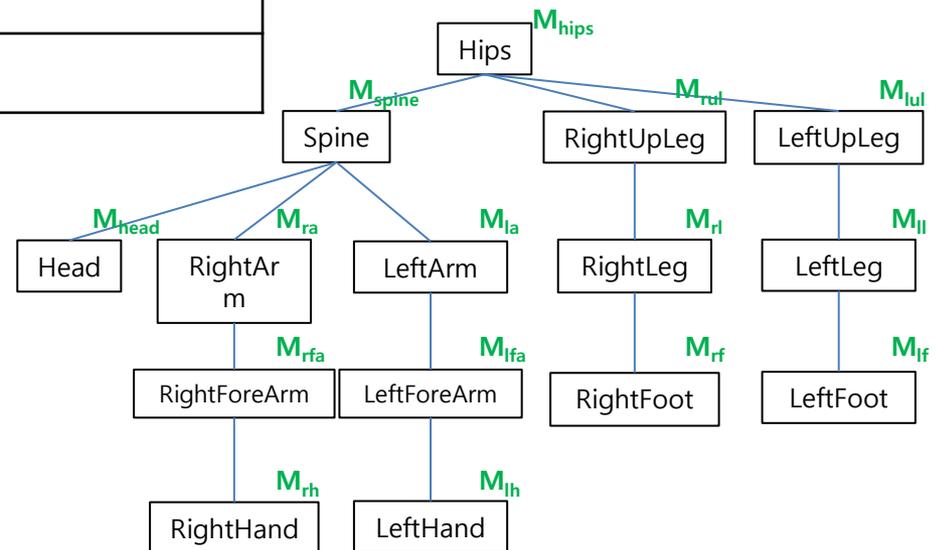
- $$\mathbf{p}_4 = \mathbf{M}_1 \mathbf{M}_2 \mathbf{M}_3 \mathbf{M}_4 \mathbf{p}_0$$

node 4's frame



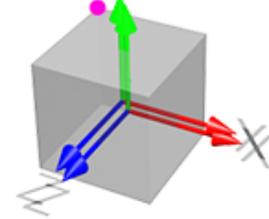
Computing Global Transform from Series of Local Transforms

Node i	Global Transform $G_i = \dots$
Hips	M_{hips}
Spine	$M_{hips} M_{spine}$
Head	$M_{hips} M_{spine} M_{head}$
RightArm	$M_{hips} M_{spine} M_{ra}$
RightForeArm	$M_{hips} M_{spine} M_{ra} M_{rfa}$
RightHand	$M_{hips} M_{spine} M_{ra} M_{rfa} M_{rh}$
LeftArm	$M_{hips} M_{spine} M_{la}$
...	



Rendering Hierarchical Models

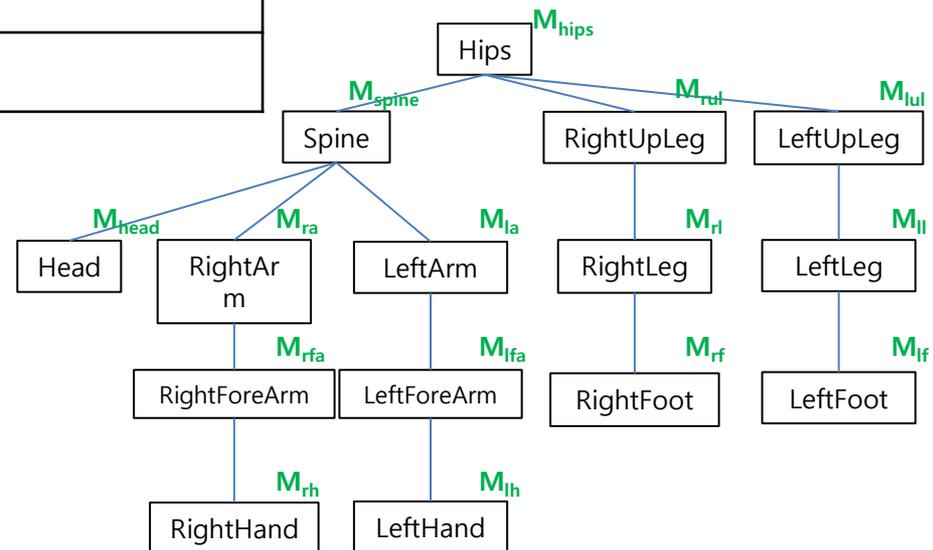
$$\mathbf{p}_0 = [-0.5, 0.5, -0.5]$$



Node i	Global Transform $G_i = \dots$
Hips	M_{hips}
Spine	$M_{hips} M_{spine}$
Head	$M_{hips} M_{spine} M_{head}$
RightArm	$M_{hips} M_{spine} M_{ra}$
RightForeArm	$M_{hips} M_{spine} M_{ra} M_{rfa}$
RightHand	$M_{hips} M_{spine} M_{ra} M_{rfa} M_{rh}$
LeftArm	$M_{hips} M_{spine} M_{la}$
...	

Let's say i -th node is rendered as a **unit cube** above (without scaling), its vertex position \mathbf{p}_i' w.r.t. world frame is...

$$\mathbf{p}_i' = \mathbf{G}_i \mathbf{p}_0$$

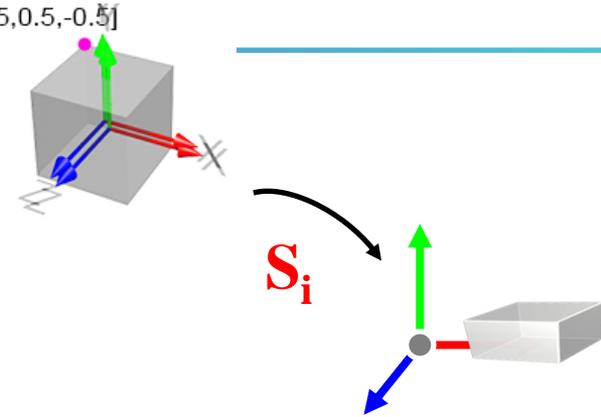


Rendering Hierarchical Models

– Shape Transformation

Node i	Global Transform $G_i = \dots$
Hips	M_{hips}
Spine	$M_{hips} M_{spine}$
Head	$M_{hips} M_{spine} M_{head}$
RightArm	$M_{hips} M_{spine} M_{ra}$
RightForeArm	$M_{hips} M_{spine} M_{ra} M_{rfa}$
RightHand	$M_{hips} M_{spine} M_{ra} M_{rfa} M_{rh}$
LeftArm	$M_{hips} M_{spine} M_{la}$
...	

$$\mathbf{p}_0 = [-0.5, 0.5, -0.5]$$



Let's say i -th node is rendered as a cuboid transformed by S_i from the unit cube, its vertex position \mathbf{p}_i' w.r.t. world frame is...

$$\mathbf{p}_i' = G_i S_i \mathbf{p}_0$$

- You might want to use "shape transformation" S_{ij} for j -th shape of i -th node.

Rendering Hierarchical Models

Node i	Global Transform $G_i = \dots$
Hips	M_{hips}
Spine	$M_{hips} M_{spine}$
Head	$M_{hips} M_{spine} M_{head}$
RightArm	$M_{hips} M_{spine} M_{ra}$
RightForeArm	$M_{hips} M_{spine} M_{ra} M_{rfa}$
RightHand	$M_{hips} M_{spine} M_{ra} M_{rfa} M_{rh}$
LeftArm	$M_{hips} M_{spine} M_{la}$
...	

- To render a hierarchical model, you can store global transform G_i for each node (i -th node) and use it for rendering.

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. **2021123456: 4.0**

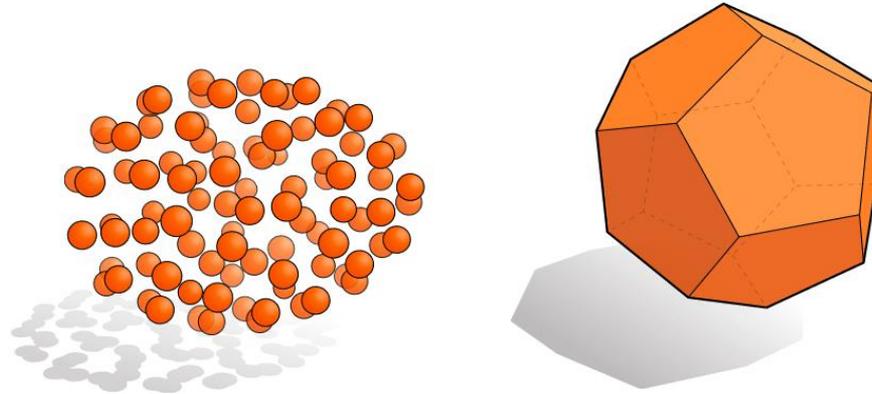
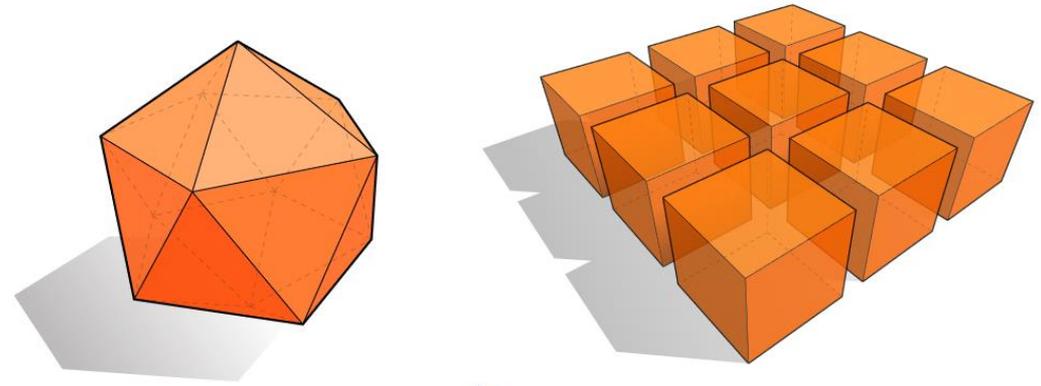
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Mesh

Many ways to digitally encode geometry

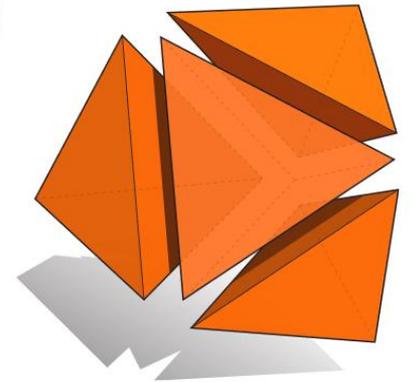
■ EXPLICIT

- point cloud
- polygon mesh
- subdivision, NURBS
- L-systems
- ...



■ IMPLICIT

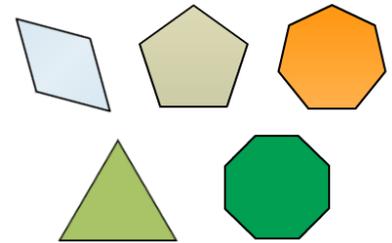
- level set
- algebraic surface
- ...



■ Each choice best suited to a different task/type of geometry

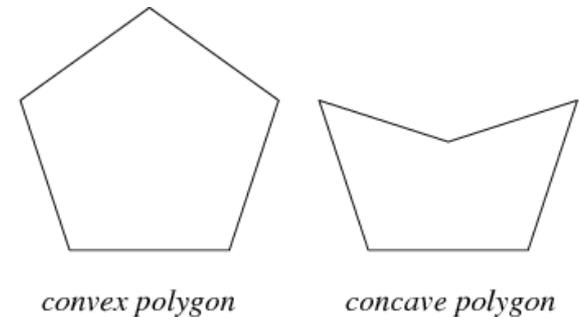
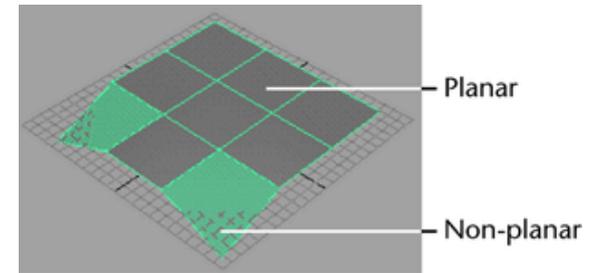
The Most Popular Representation : Polygon Mesh

- Because this can model any arbitrary complex shapes with relatively simple representations and can be rendered fast.
- **Polygon:** a “closed” shape with straight sides
- **Polygon mesh:** a bunch of polygons in 3D space that are connected together to form a surface
 - Usually use *triangles* or *quads* (4 side polygon)



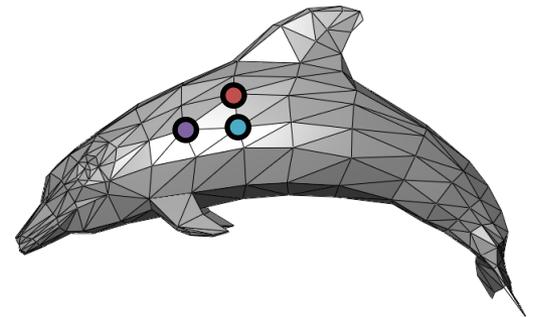
Triangle Mesh

- A general N-polygon can be
 - Non-planar
 - Non-convex
- , which are not desirable for fast rendering.
- A triangle does not have such problems. It's always planar & convex.
- and N-polygons can be composed of multiple triangles.
- That's why modern GPUs draw everything as a set of triangles.
- So, we'll focus on triangle meshes.



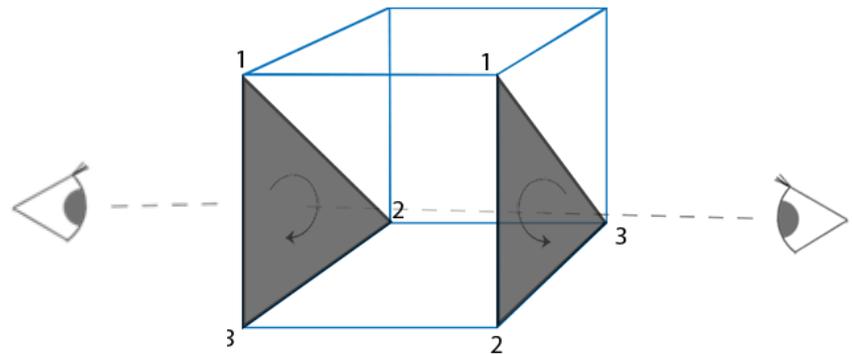
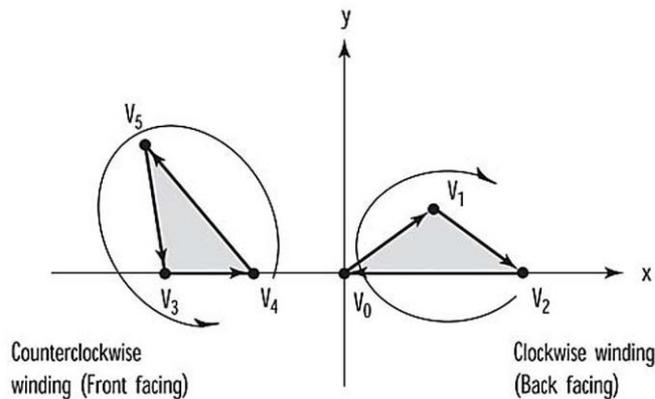
Representation for Triangle Mesh

- It's about how to store
 - vertex positions
 - relationship between vertices (to make triangles)
- on memory.
- Two basic representations:
 - Separate triangles
 - Indexed triangle set



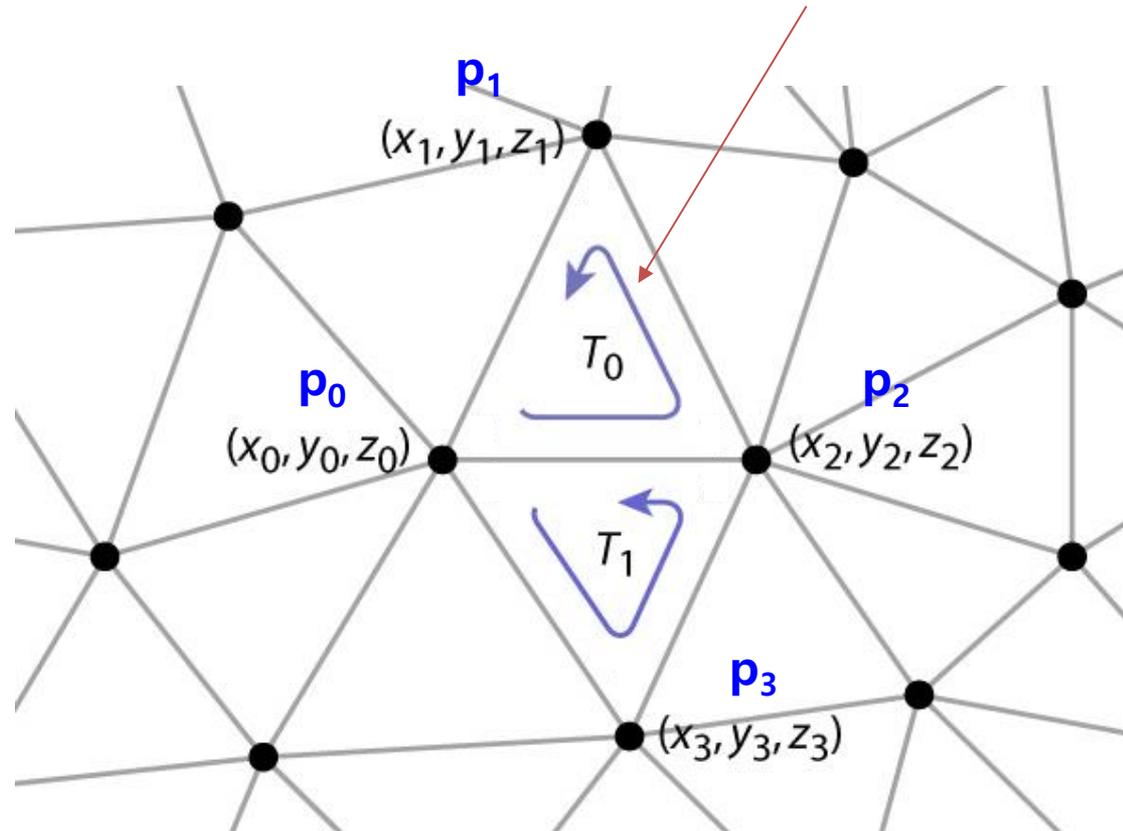
Vertex Winding Order

- *Vertex winding order* is the order in which the vertices of a polygon are listed in a representation of a polygon.
- Determines which side of the polygon is "front".
 - In OpenGL, by default, polygons whose vertices appear in counterclockwise (CCW) order on the screen is front-facing.
 - In Direct3D, the default front-facing winding order is clockwise (CW).



Separate triangles

counter-clockwise order

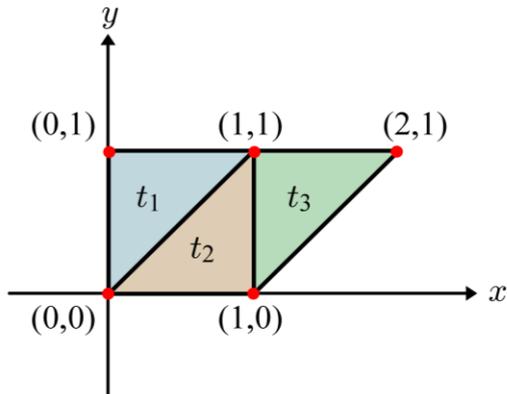


vertex array

	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	\vdots	\vdots	\vdots

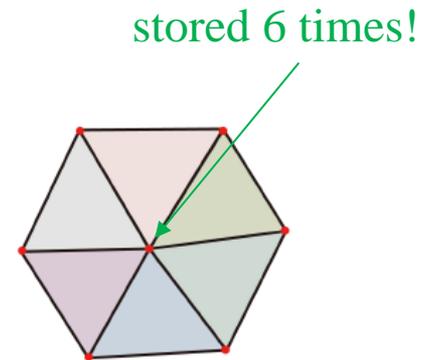
Separate Triangles

- Limitations
 - Wastes memory space
 - Cracks due to roundoff
 - Difficulty of finding neighbor triangles
 - If you want find "neighbor" triangles of t_2 , you have to find all "zero-distance" vertices from t_2 's each vertex.



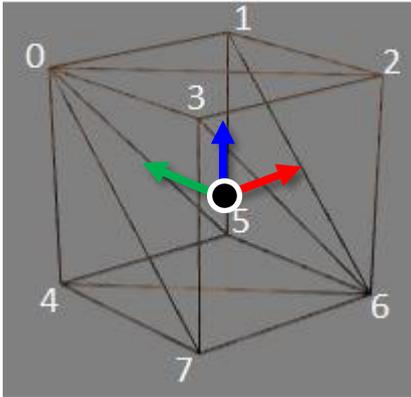
vertex buffer

(0,1)	t ₁
(0,0)	
(1,1)	t ₂
(0,0)	
(1,0)	t ₃
(1,1)	
(1,0)	
(2,1)	



(1,1) is stored 3 times!

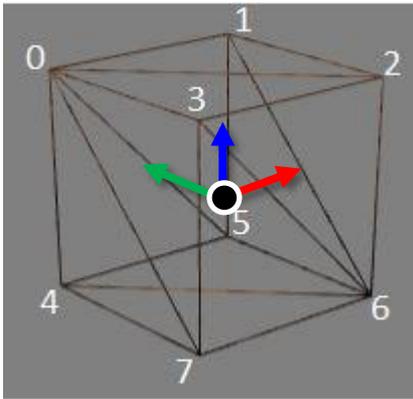
Example: a cube of length 2



vertex index	position
0	(-1 , 1 , 1)
1	(1 , 1 , 1)
2	(1 , -1 , 1)
3	(-1 , -1 , 1)
4	(-1 , 1 , -1)
5	(1 , 1 , -1)
6	(1 , -1 , -1)
7	(-1 , -1 , -1)

Example Cube in Separate Triangles

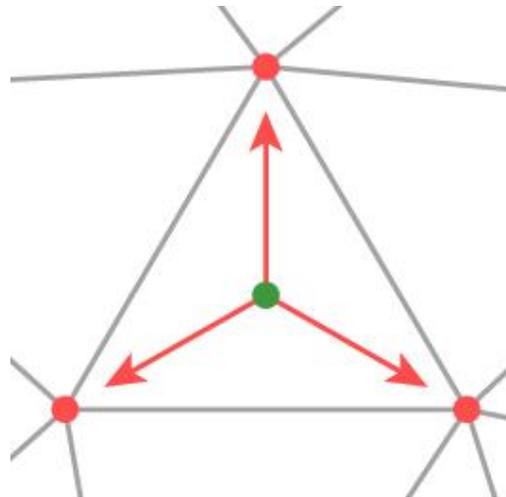
- In separate triangles scheme, the cube is represented by the positions of the 36 vertices that make up its 12 triangles.



```
# vertex array
# triangle 0
-1 , 1 , 1, # v0
 1 , -1 , 1, # v2
 1 , 1 , 1, # v1
# triangle 1
-1 , 1 , 1, # v0
-1 , -1 , 1, # v3
 1 , -1 , 1, # v2
# triangle 2
-1 , 1 , -1, # v4
 1 , 1 , -1, # v5
 1 , -1 , -1, # v6
# triangle 3
-1 , 1 , -1, # v4
 1 , -1 , -1, # v6
-1 , -1 , -1, # v7
# triangle 4
-1 , 1 , 1, # v0
 1 , 1 , 1, # v1
 1 , 1 , -1, # v5
# triangle 5
-1 , 1 , 1, # v0
 1 , 1 , -1, # v5
-1 , 1 , -1, # v4
# triangle 6
-1 , -1 , 1, # v3
 1 , -1 , -1, # v6
 1 , -1 , 1, # v2
# triangle 7
-1 , -1 , 1, # v3
-1 , -1 , -1, # v7
 1 , -1 , -1, # v6
# triangle 8
 1 , 1 , 1, # v1
 1 , -1 , 1, # v2
 1 , -1 , -1, # v6
# triangle 9
 1 , 1 , 1, # v1
 1 , -1 , -1, # v6
 1 , 1 , -1, # v5
# triangle 10
-1 , 1 , 1, # v0
-1 , -1 , -1, # v7
-1 , -1 , 1, # v3
# triangle 11
-1 , 1 , 1, # v0
-1 , 1 , -1, # v4
-1 , -1 , -1, # v7
```

Indexed triangle set

- Store each vertex once
- Each triangle "points to" its three vertices



Indexed triangle set

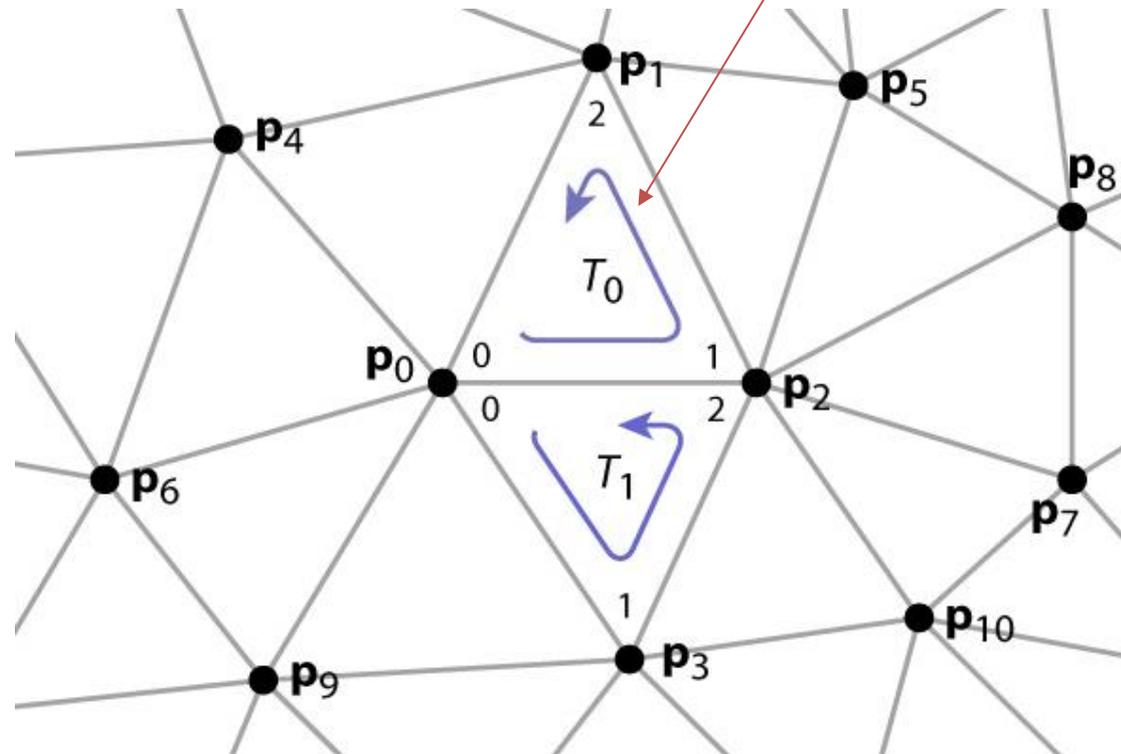
counter-clockwise order

vertex array

verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
	\vdots

index array

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	\vdots

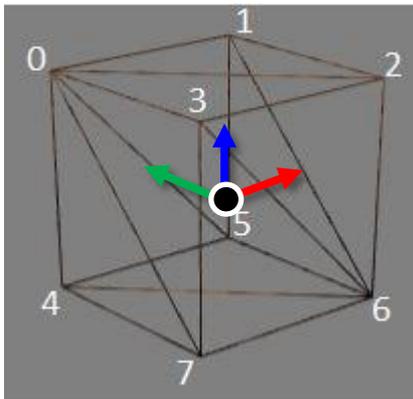


Indexed Triangle Set

- Memory efficient: Each vertex position is stored only once.
- Represents topology and geometry separately.
 - Topology: How vertices are connected to form triangles.
 - Geometry: The actual positions of the vertices in space.
- Finding neighbor triangles is at least well defined.
 - Neighbor triangles share same vertex indices.

Example Cube in Indexed Triangle Set

- In indexed triangle set scheme, the cube is represented by the **positions of its 8 vertices** and the **vertex indices of its 12 triangles**.



```
# vertex array
-1 , 1 , 1 , # v0
 1 , 1 , 1 , # v1
 1 , -1 , 1 , # v2
-1 , -1 , 1 , # v3
-1 , 1 , -1 , # v4
 1 , 1 , -1 , # v5
 1 , -1 , -1 , # v6
-1 , -1 , -1 , # v7
```

```
# index array
0 , 2 , 1 , # t0
0 , 3 , 2 , # t1
4 , 5 , 6 , # t2
4 , 6 , 7 , # t3
0 , 1 , 5 , # t4
0 , 5 , 4 , # t5
3 , 6 , 2 , # t6
3 , 7 , 6 , # t7
1 , 2 , 6 , # t8
1 , 6 , 5 , # t9
0 , 7 , 3 , # t10
0 , 4 , 7 , # t11
```

Quiz 2

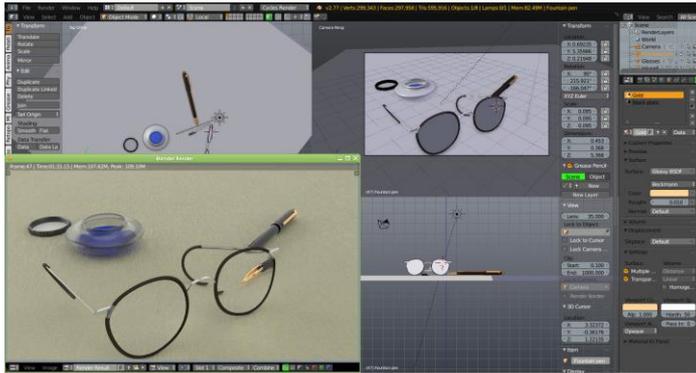
- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"

- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2021123456: 4.0**

- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Creating Polygon Meshes

- Usually, polygon meshes are created using 3D modeling programs.
 - A file that stores polygon mesh data is called an *object file* or *model file*.



Blender



Maya

- Applications (such as games) usually load vertex and index data from an *object file* and draw the object using the loaded data.

3D Model File Formats

- DXF – AutoCAD
 - Supports 2-D and 3-D; binary
- 3DS – 3DS MAX
 - Flexible; binary
- VRML – Virtual reality modeling language
 - ASCII – Human readable (and writeable)
- **OBJ – Wavefront OBJ format**
 - ASCII – Human readable (and writeable)
 - Extremely simple
 - Widely supported
- Let's take a closer look at OBJ format!

OBJ File Format

```
# this is a comment

# List of vertex positions, in (x, y, z) form.
v 0.123 0.234 0.345
v 0.2 0.5 0.3
v ...
...

# List of vertex normals, in (x,y,z) form; normals
might not be unit vectors.
vn 0.707 0.000 0.707
vn ...
...

# List of vertex texture coordinates, in (u, v) form.
vt 0.500 1
vt ...
...
```

OBJ File Format

```
# List of faces (all argument indices are 1-based indices!)

# with vertex positions only - vertex_position_index
f 1 2 3
f 2 3 4
...

#
vertex_position_index/texture_coordinates_index/vertex_normal_index
f 6/4/1 3/5/3 7/6/5

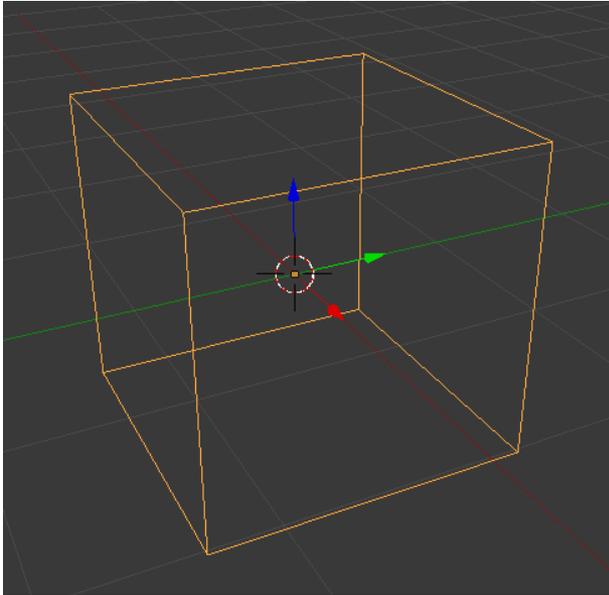
# vertex_position_index//vertex_normal_index
f 7//1 8//2 9//3
...

# vertex_position_index/texture_coordinates_index
f 3/1 4/2 5/3
...
```

OBJ File Format

- Other supported features:
 - for polyline
 - 1 5 8 1 2 4 9
 - for materials
 - `mtllib [external .mtl file name]`
 - `usemtl [material name]`
 - ...
- You don't need to use these features in this class.

An OBJ Example



```
# A simple cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000000
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

OBJ Sources

- <https://free3d.com/>
- <https://www.cgtrader.com/free-3d-models>
- You can download any .obj model files from these sites and open them in Blender.
- OBJ file format is very popular:
 - Most modeling programs will export OBJ files
 - Most rendering packages will read in OBJ files

Lab Session

- Now, let's start the lab today.