
Computer Graphics

7 - Hierarchical Modeling, Mesh

Yoonsang Lee
Hanyang University

Spring 2026

(Updated) Midterm Exam Announcement

- Date & time: **May 4 (Mon), 10:00 - 11:30 AM**
- Place: IT.BT 508
- Scope: Lecture, Lab, Lab Check 2~7, **Project 1**
- Exam format: **Refer to past midterm and final exam problems** from previous Computer Graphics courses: <https://cgrhyu.github.io/3-courses.html>
- **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!).
 - This policy is to prevent early finishers from sharing exam information with late arrivals.
- Bring your **student ID card** to the exam.

(Updated) Midterm Exam Announcement

- **No digital or electronic devices** are allowed during the exam.
- This includes (but is not limited to): smartphones, smartwatches, tablets, laptops, wireless earphones, and smart glasses.
- All devices must be **powered off and kept in your bag**.
- **Do not wear any electronic devices** during the exam.
- Use of a device will be treated as **academic misconduct**.
- If you are unsure whether a device is allowed, **do not bring it**.

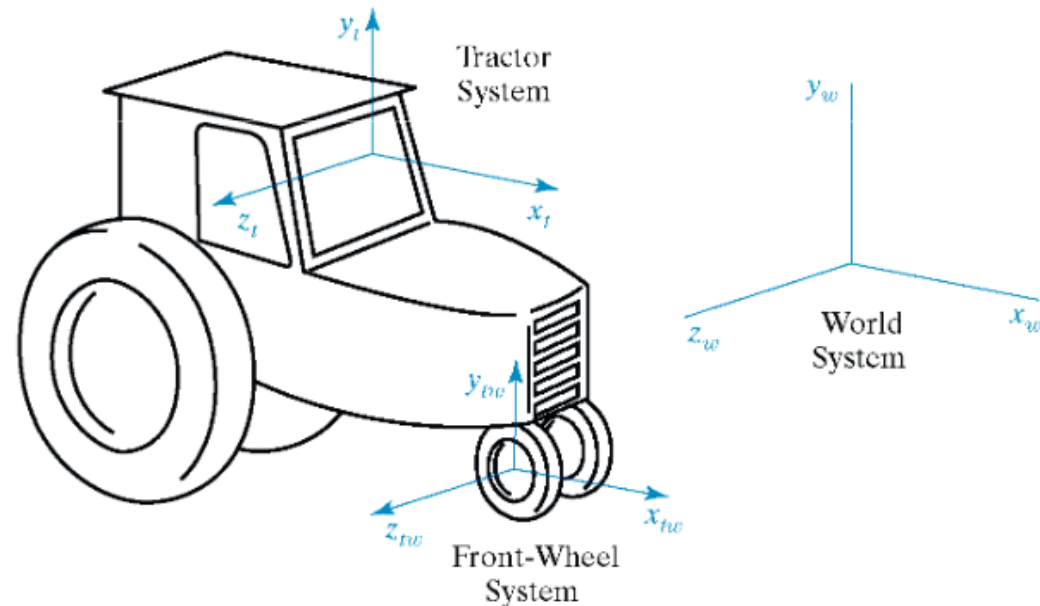
Outline

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

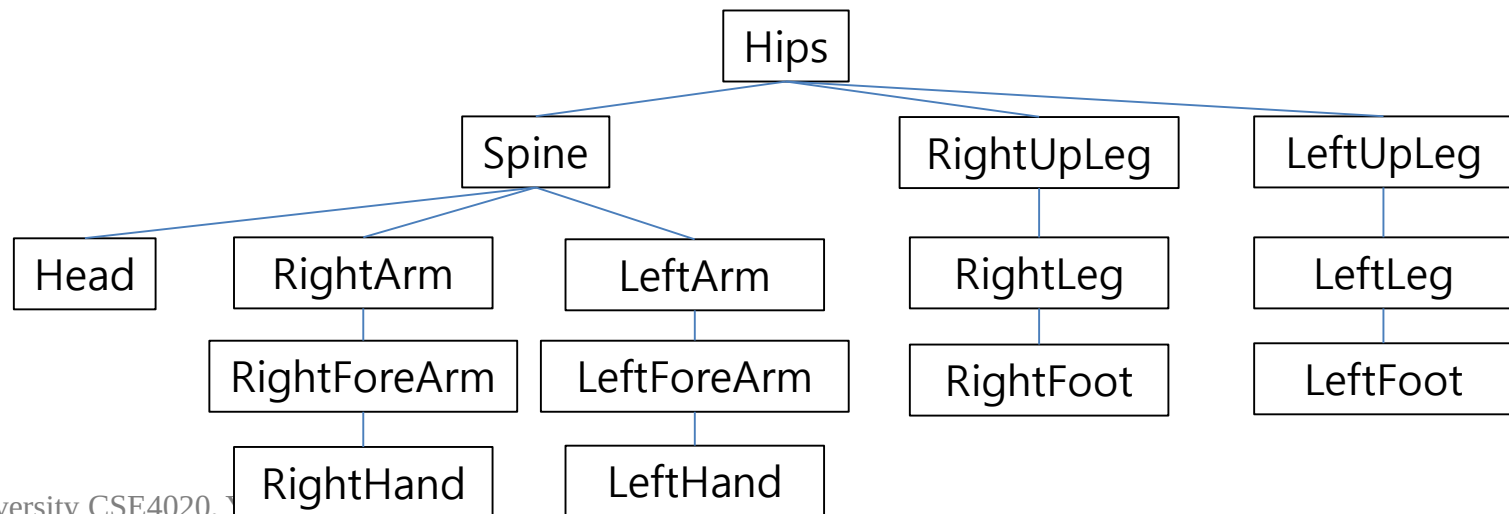
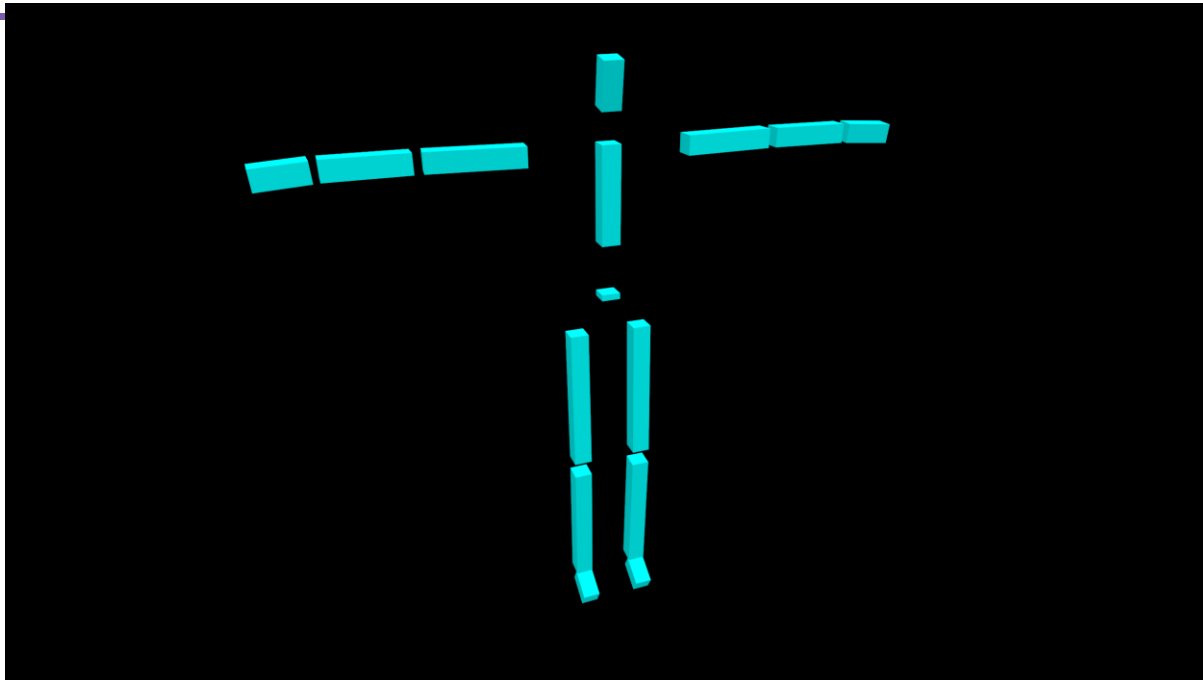
Hierarchical Modeling

Hierarchical Modeling

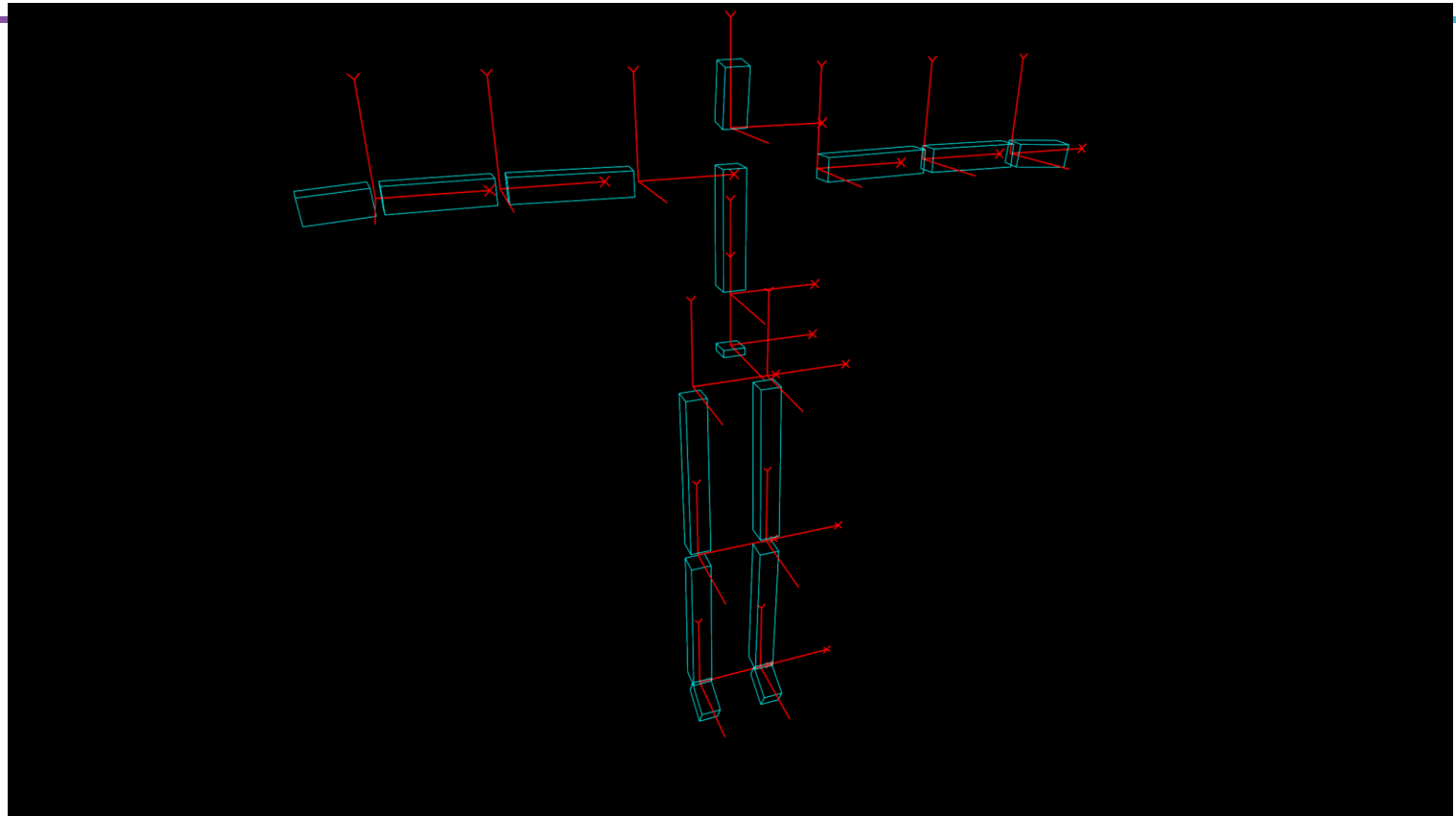
- Organize parts in a tree structure: **parent parts contain child parts**
 - Each part has its own body frame.
 - Each part's movement is described w.r.t. its parent's body frame.



Example - Human Figure

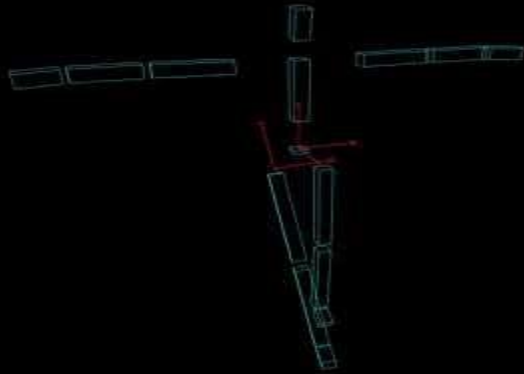


Human Figure - Frames

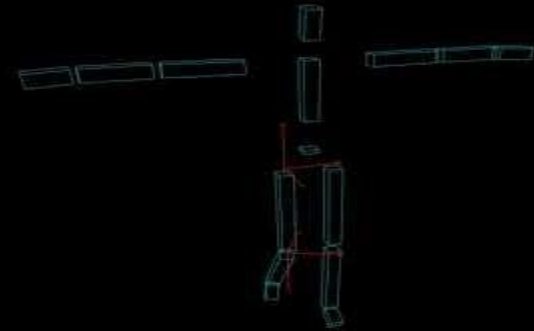


- Each part has its own body frame.

Human Figure - Movement of rhip & rknee



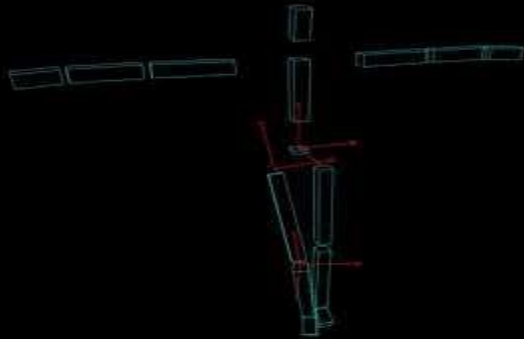
<https://youtu.be/Q7lhvMkCSCg>



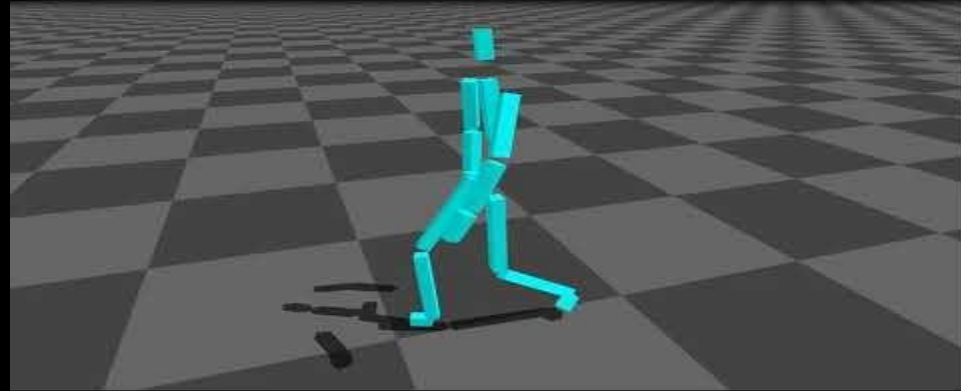
<https://youtu.be/Q5R8WGUwpFU>

- Each part's movement is described w.r.t. its parent's frame.
→ Each part has its **own transformation** relative to its parent.
- Moving a parent moves all its children (grouping effect)
→ Makes complex motion easy to control

Human Figure - Movement of more joints



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

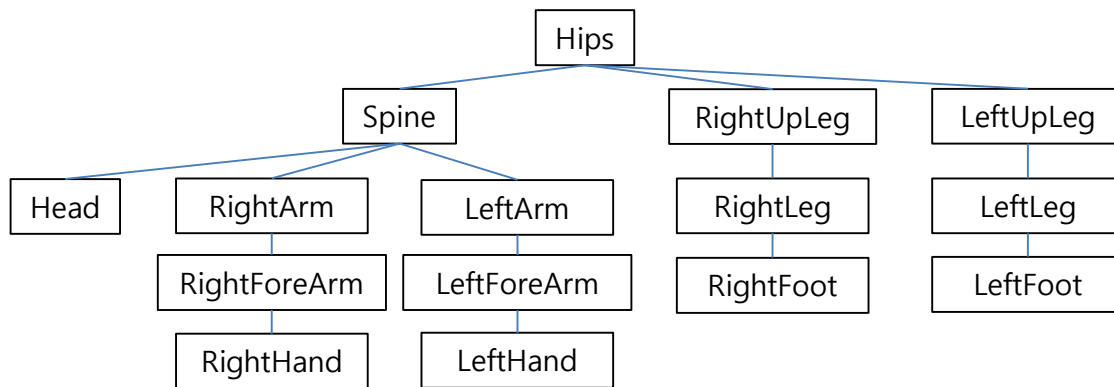


<https://youtu.be/PEhyWI8LGBY>

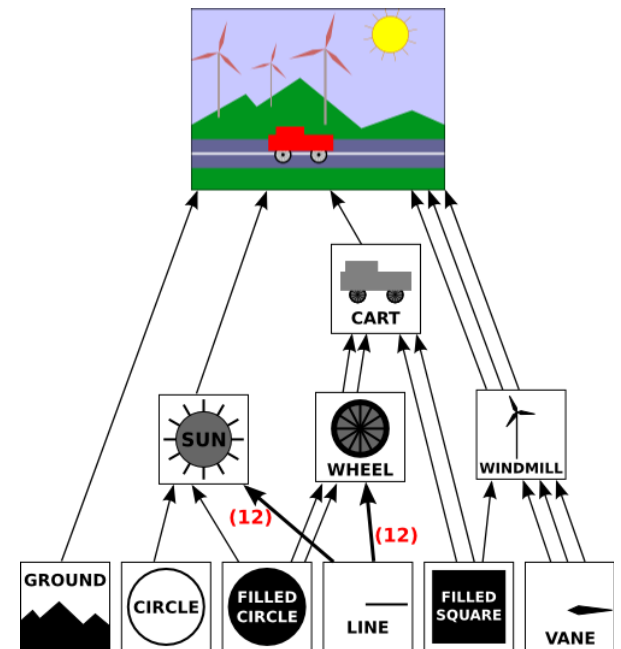
- Each part's movement is described w.r.t. its parent's frame.
→ Each part has its **own transformation** relative to its parent.
- Moving a parent moves all its children (grouping effect)
→ Makes complex motion easy to control

Hierarchical Model

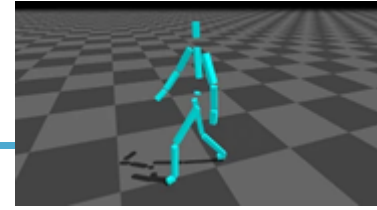
- Each node has its **own transformation** relative to its parent
 - Hierarchical models are represented as **tree structures**
- Another example: scene graph (represents a full scene)



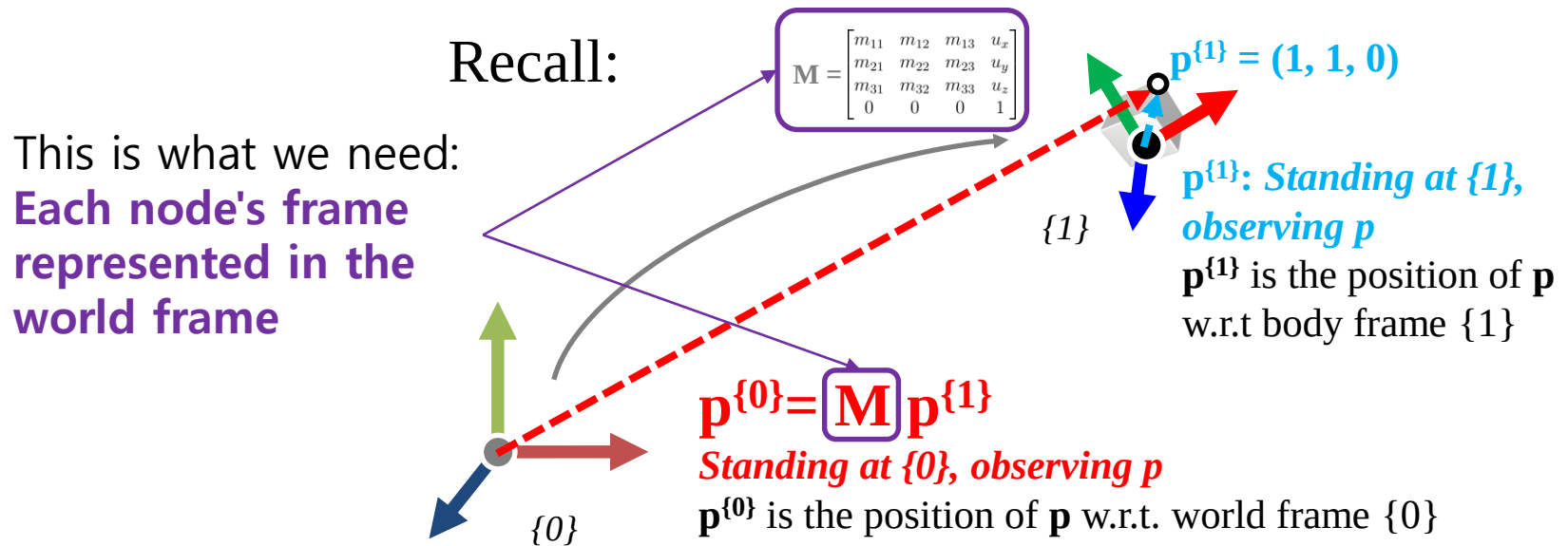
Scene graph example



Rendering Hierarchical Models



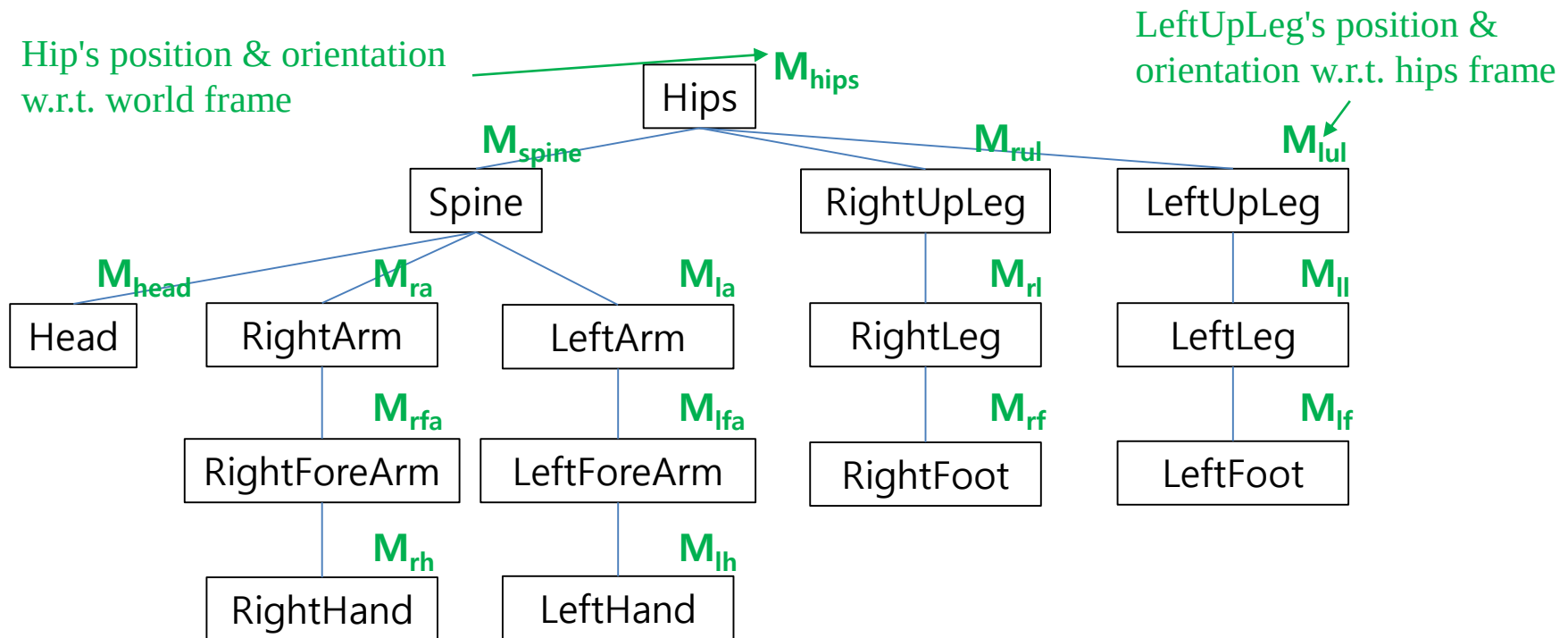
- To render, we must express every point in the world frame
- → “Changing from **Body Frame** to **World Frame**”



- In hierarchy, this is applied recursively along the tree

Local Transformation

- Each node has its **own transformation** relative to its parent.
- **Local transformation**

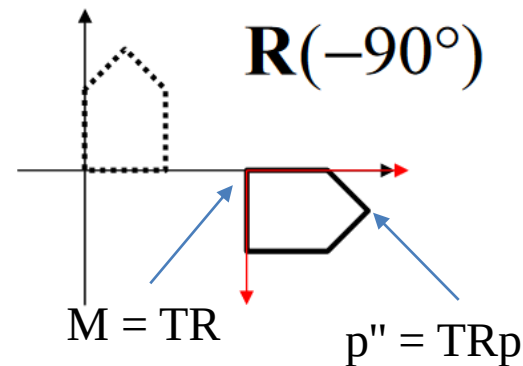
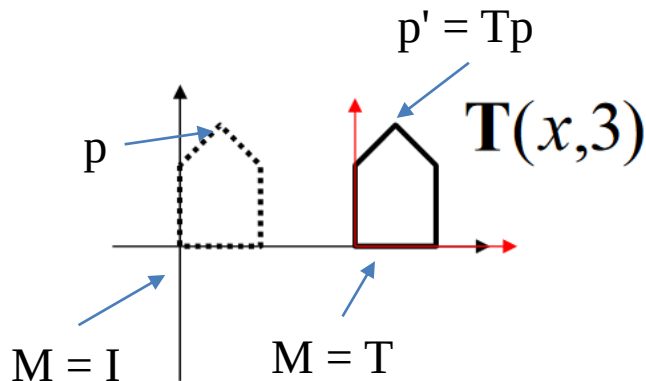


Global Transformation

- **Each node's frame** must be represented in the world frame.
- → **Global transformation**
- How do we compute the **global transform** of a node from the **local transforms** of its parent?

Recall: Post-(right) Multiplication

- $p' = M_1 M_2 p$ (**post-multiplication by M_2**)
 - (L-to-R)
 - 1') Apply M_1 w.r.t. **body frame I** (world frame) to **update body frame to M_1**
 - 2') Apply M_2 w.r.t. **body frame M_1** to **update body frame to $M_1 M_2$**
 - 3') Locate p in body frame $M_1 M_2$

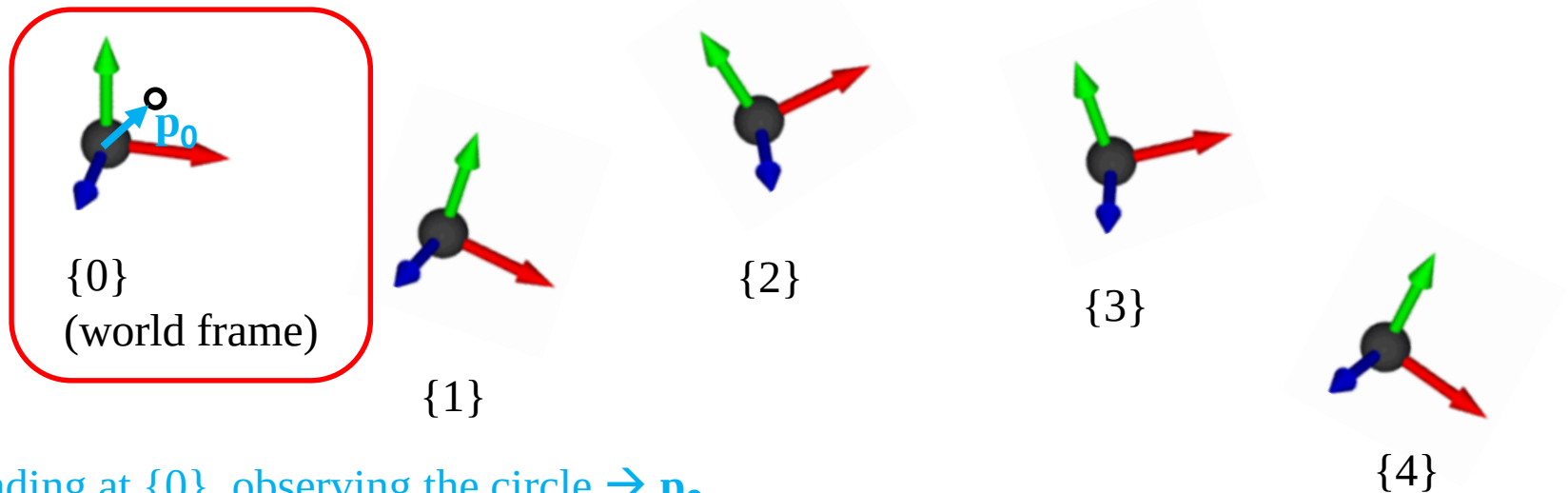


(M defines the **body frame** w.r.t. world frame)

Interpretation of a Series of Transformations

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

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

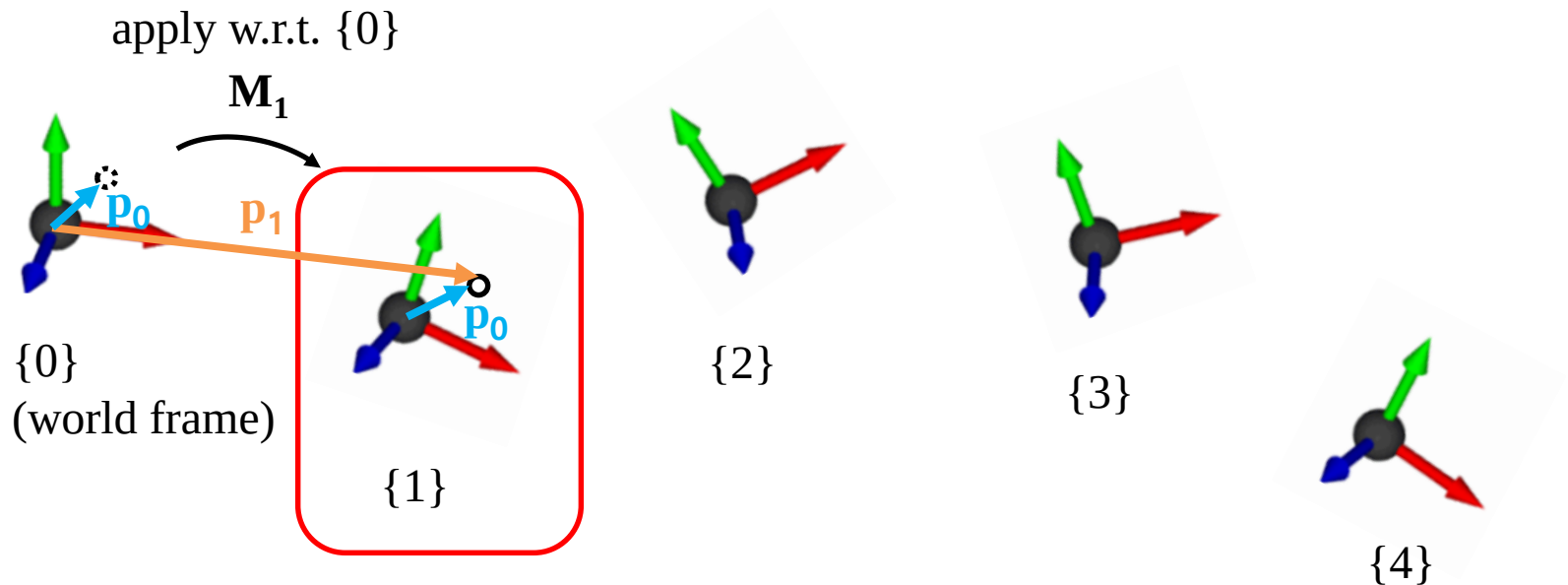


Standing at $\{0\}$, observing the circle $\rightarrow \mathbf{p}_0$

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}



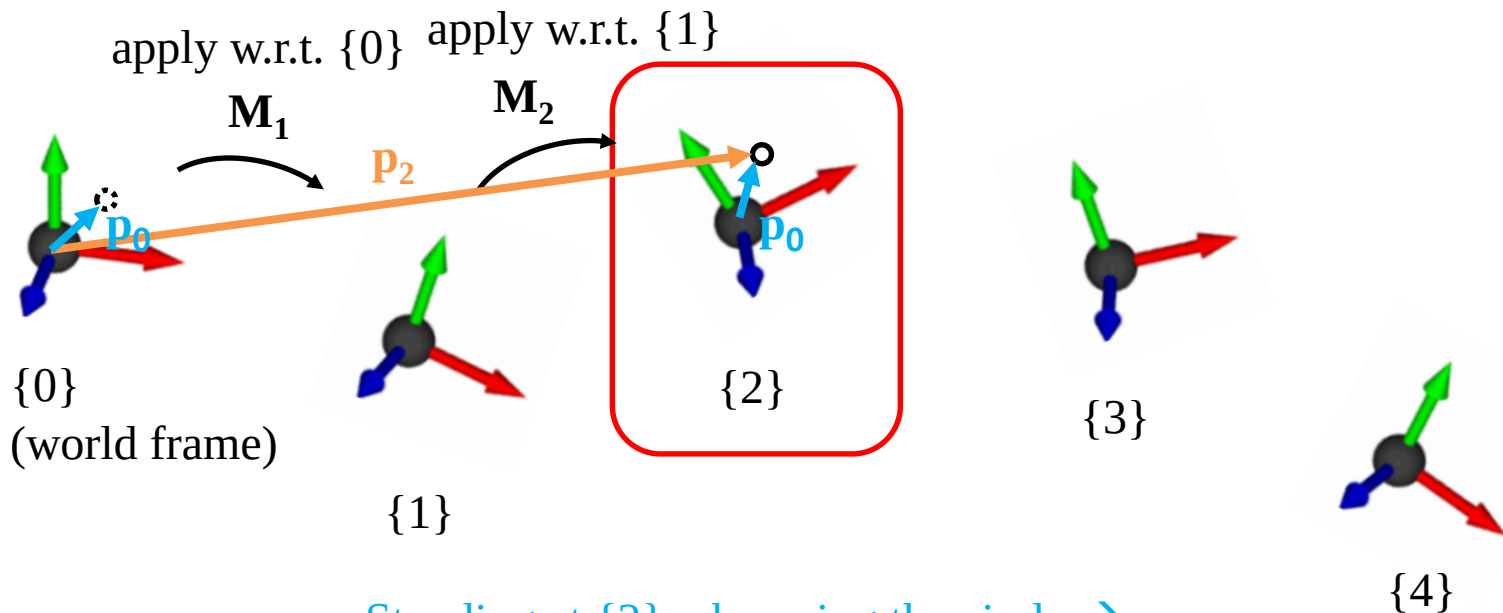
Standing at {1}, observing the circle $\rightarrow p_0$

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}

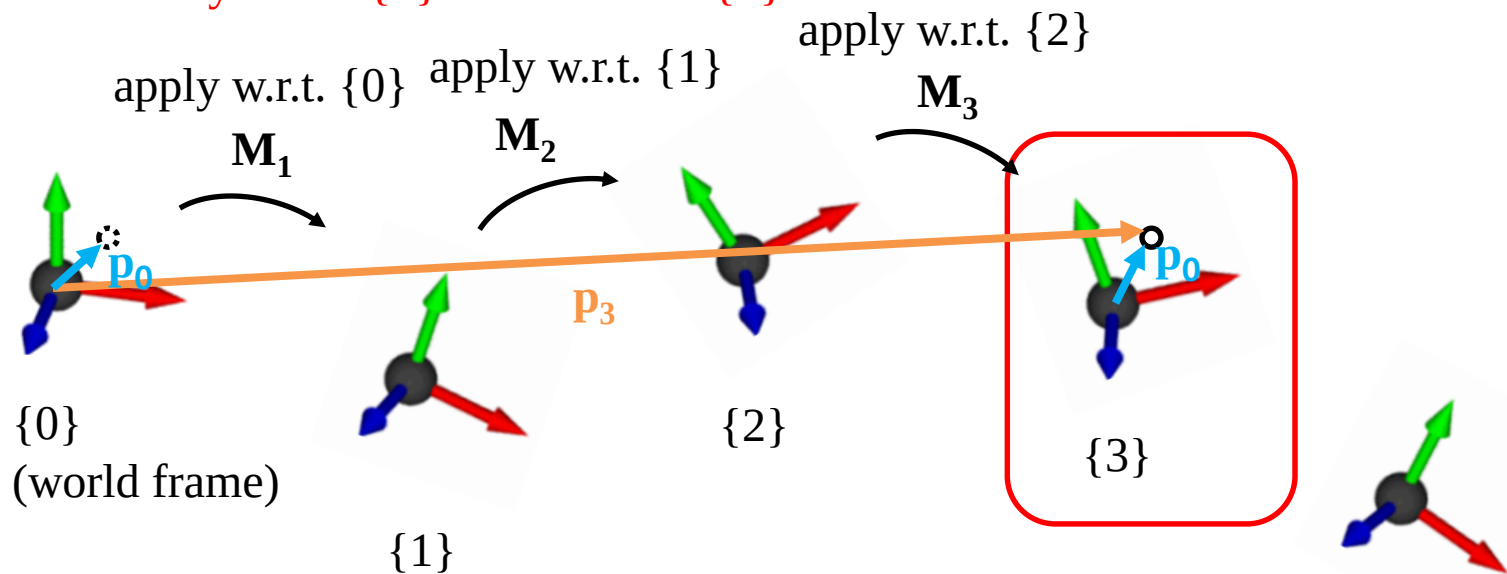


Standing at {2}, observing the circle $\rightarrow \mathbf{p}_0$
Standing at {0}, observing the circle $\rightarrow \mathbf{p}_2$

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}

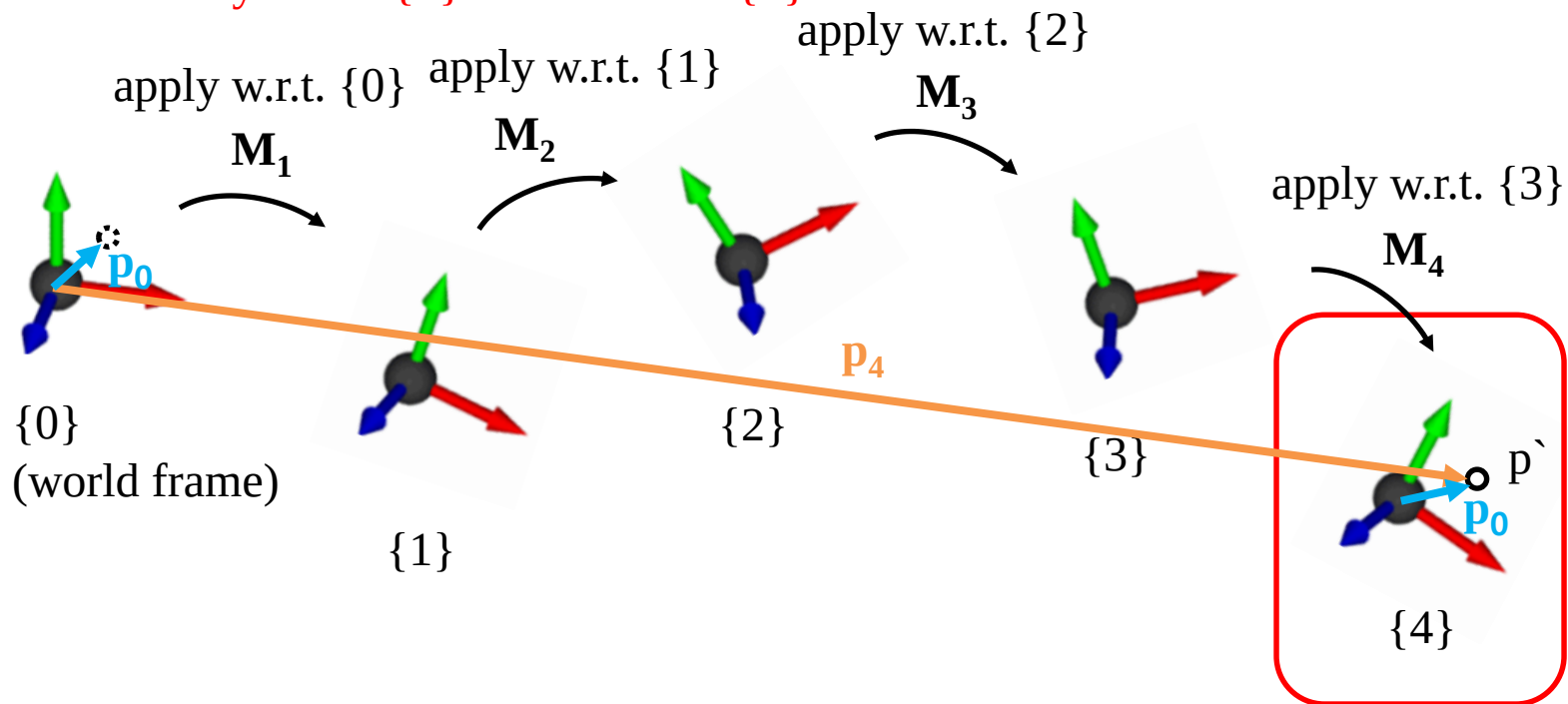


Standing at {3}, observing the circle $\rightarrow \mathbf{p}_0$ {4}
Standing at {0}, observing the circle $\rightarrow \mathbf{p}_3$

Interpretation of a Series of Transformations

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

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



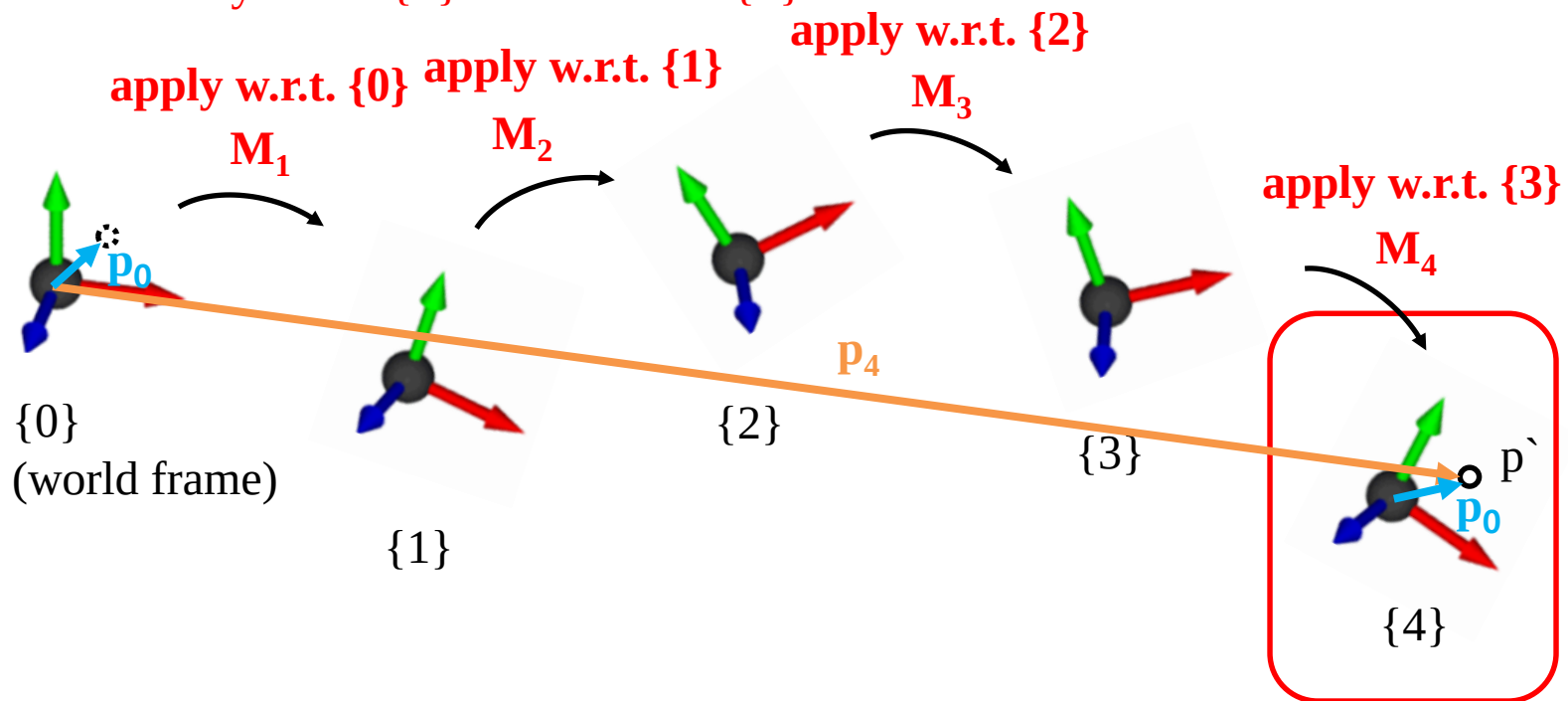
Standing at {4}, observing the circle $\rightarrow \mathbf{p}_0$

Standing at {0}, observing the circle $\rightarrow \mathbf{p}_4$

Interpretation of a Series of Transformations

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

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



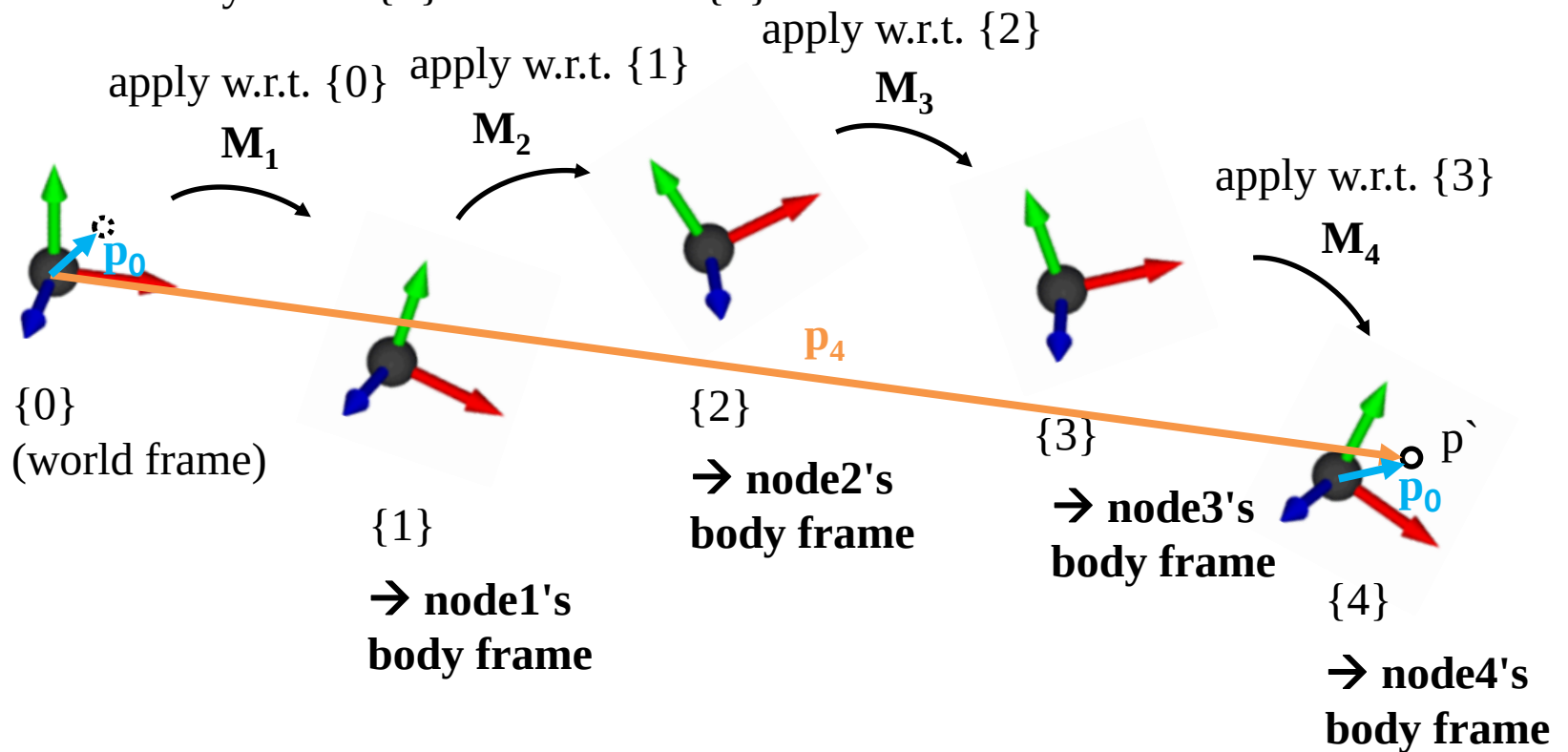
Standing at {4}, observing the circle $\rightarrow \mathbf{p}_0$

Standing at {0}, observing the circle $\rightarrow \mathbf{p}_4$

Interpretation of a Series of Transformations

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

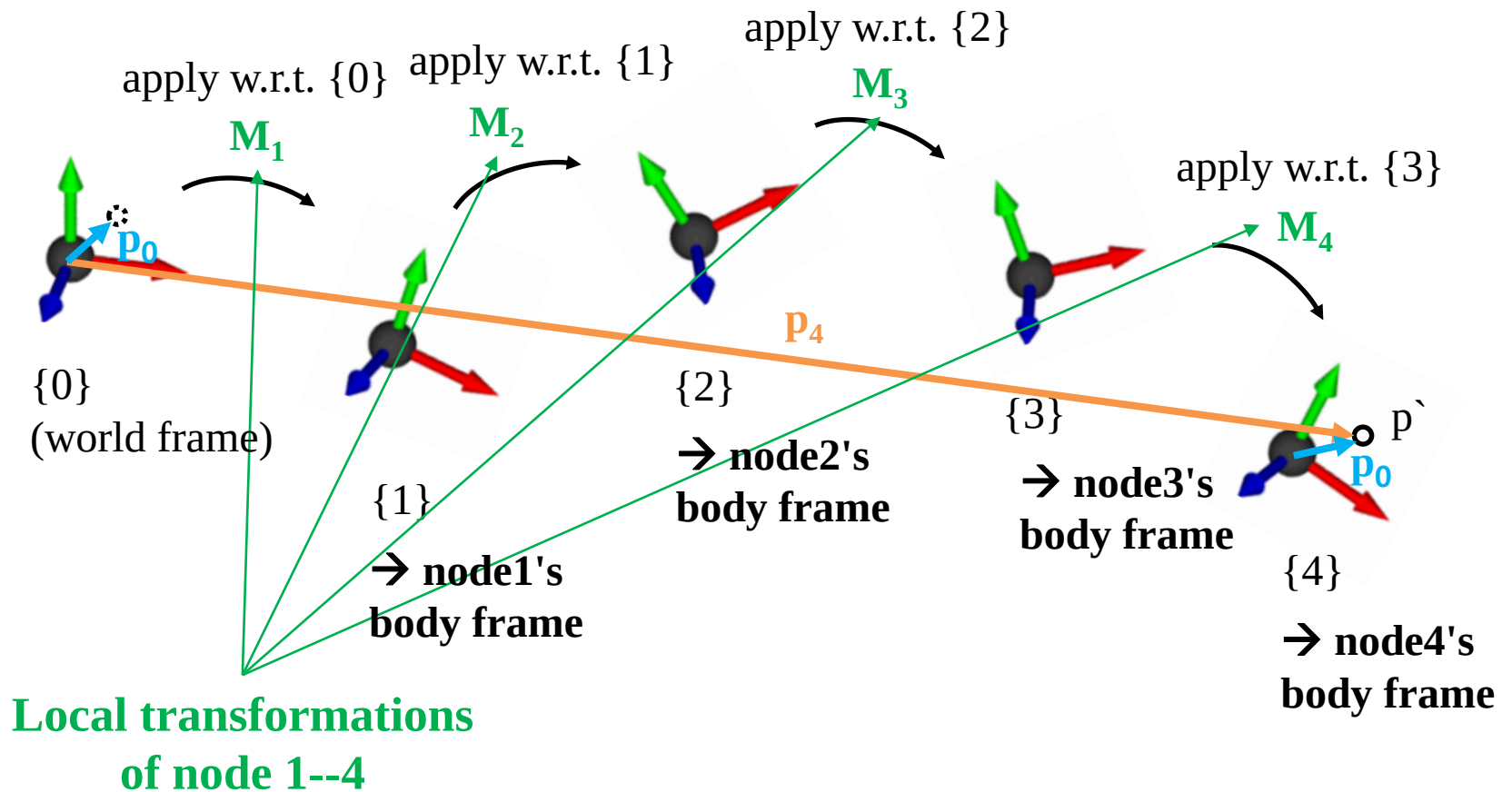
Current body frame {4} defined w.r.t. {0}



Computing Global Transform from Series of Local Transforms

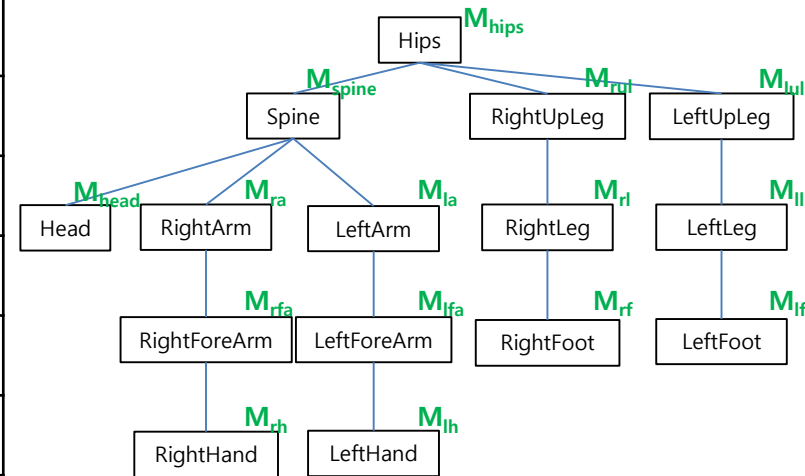
- $$\mathbf{p}_4 = \underbrace{\mathbf{M}_1 \mathbf{M}_2 \mathbf{M}_3 \mathbf{M}_4}_{\text{node 4's frame}} \mathbf{p}_0$$

Global transformation of node 4



Computing Global Transform from Series of Local Transforms

Node i	Global Transform G_i
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	$M_{hips} M_{spine} M_{la}$
...	



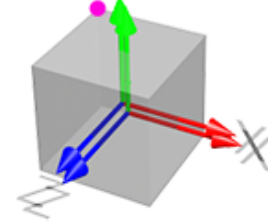
Global transformation of node i (G_i)

= product of **local transforms** along the path **from root to node i**

= **parent's global transform** * **node i's local transform**

Rendering Hierarchical Models

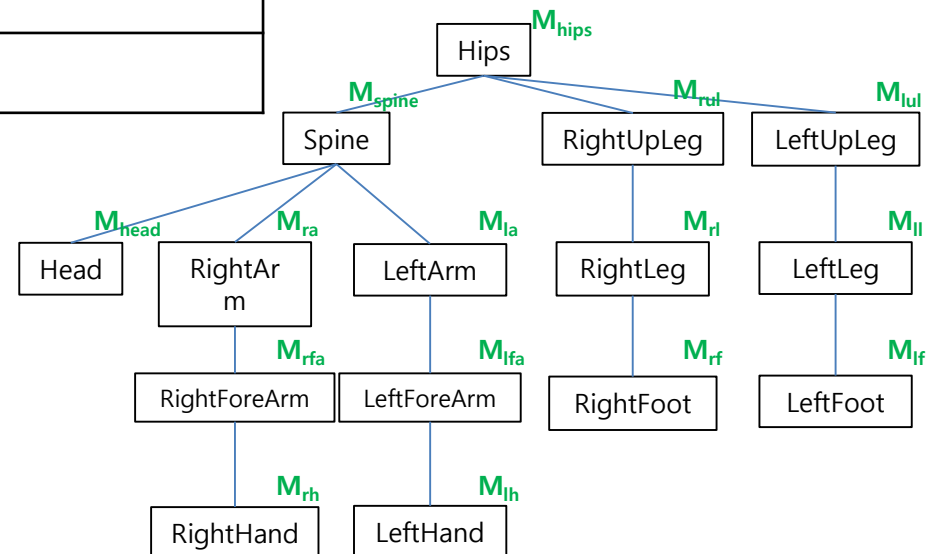
$$\mathbf{p}_0 = [-0.5, 0.5, -0.5]$$



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...

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

Node i	Global Transform $\mathbf{G}_i = \dots$
Hips	$\mathbf{M}_{\text{hips}}$
Spine	$\mathbf{M}_{\text{hips}} \mathbf{M}_{\text{spine}}$
Head	$\mathbf{M}_{\text{hips}} \mathbf{M}_{\text{spine}} \mathbf{M}_{\text{head}}$
RightArm	$\mathbf{M}_{\text{hips}} \mathbf{M}_{\text{spine}} \mathbf{M}_{\text{ra}}$
RightForeArm	$\mathbf{M}_{\text{hips}} \mathbf{M}_{\text{spine}} \mathbf{M}_{\text{ra}} \mathbf{M}_{\text{rfa}}$
RightHand	$\mathbf{M}_{\text{hips}} \mathbf{M}_{\text{spine}} \mathbf{M}_{\text{ra}} \mathbf{M}_{\text{rfa}} \mathbf{M}_{\text{rh}}$
LeftArm	$\mathbf{M}_{\text{hips}} \mathbf{M}_{\text{spine}} \mathbf{M}_{\text{la}}$
...	

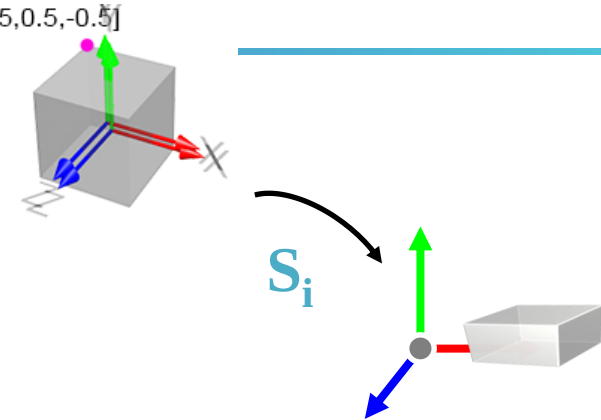


Rendering Hierarchical Models

– 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	$M_{hips} M_{spine} M_{la}$
...	

$$p_0 = [-0.5, 0.5, -0.5]$$



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

$$p_i' = G_i S_i p_0$$

- G_i : places the node in the world (global transformation)
- S_i : defines the shape in the node's local frame (shape transformation)
 - A node may have multiple shapes (S_{ij})

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	$M_{hips} M_{spine} M_{la}$
...	

- Once G_i is computed, any vertex of the node can be transformed to world space.
- Therefore, we can store G_i for each node and use it for rendering.

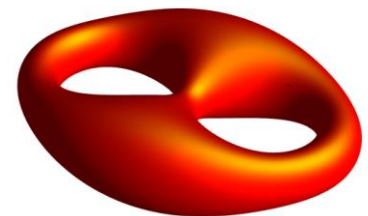
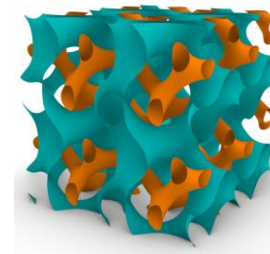
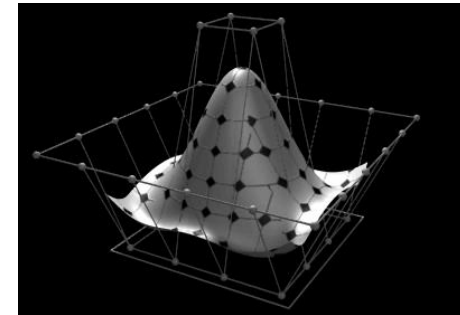
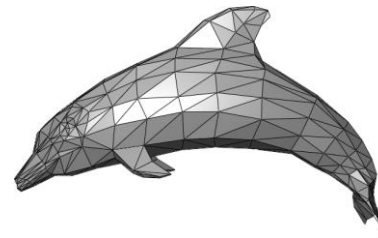
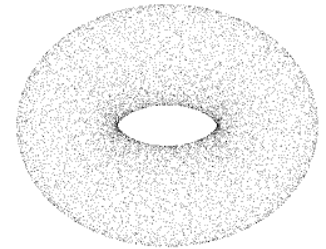
Quiz 1

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Mesh

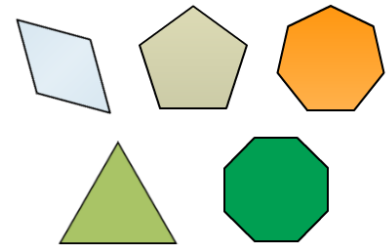
How do we represent geometry?

- There are many ways to represent geometry:
- Explicit representations
 - point clouds
 - polygon meshes
 - parametric surfaces (e.g., NURBS)
 - ...
- Implicit representations
 - level sets
 - algebraic surfaces
 - ...
- Different representations are suited for different tasks.



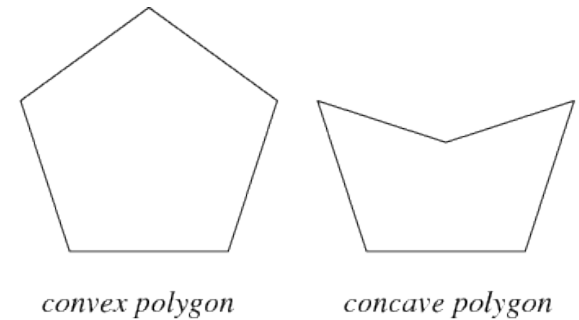
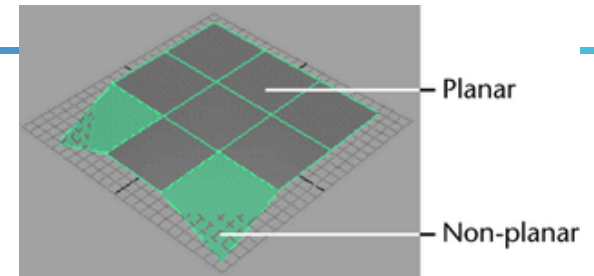
The Most Popular Representation : Polygon Mesh

- Can represent complex shapes with a simple structure
- Efficient for rendering
- **Polygon**: a closed shape with straight edges
- **Polygon mesh**
 - Connected polygons forming a surface
 - Smooth surfaces are approximated using many small polygons
 - Typically triangles or quads (four-sided polygons)



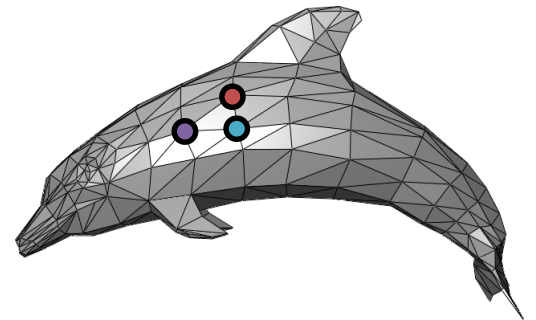
Triangle Mesh

- A general N-polygon can be
 - Non-planar
 - Non-convex
- These are not desirable for fast rendering.
- A triangle is always:
 - planar
 - convex
- Any polygon can be decomposed into triangles
- Modern GPUs render everything as triangles.
- So, we focus on triangle meshes.



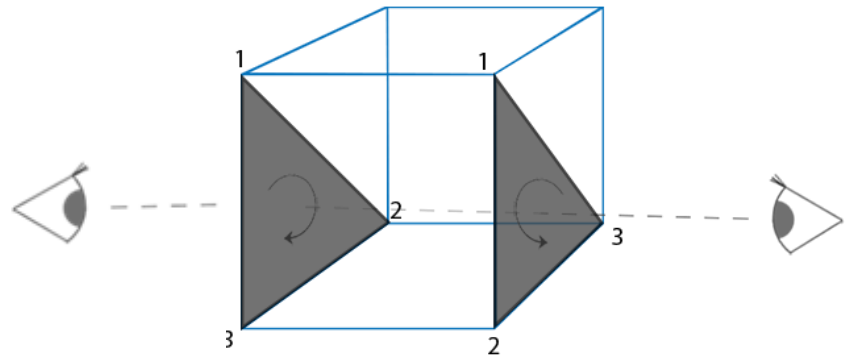
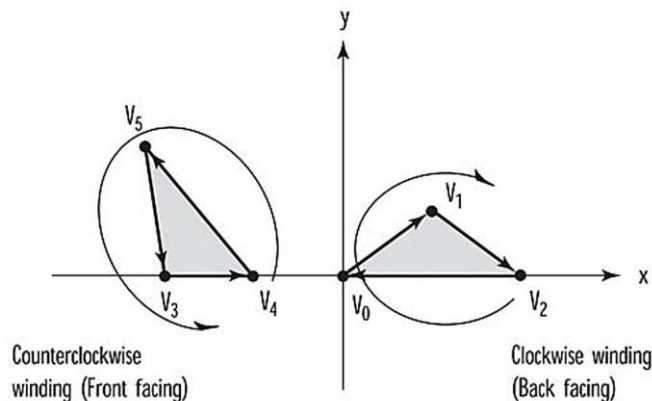
Representation for Triangle Mesh

- What do we need to store in memory?
 - Vertex positions
 - Connectivity (which vertices form triangles)
- Two basic representations:
 - Separate triangles
 - Indexed triangle set



Vertex Winding Order

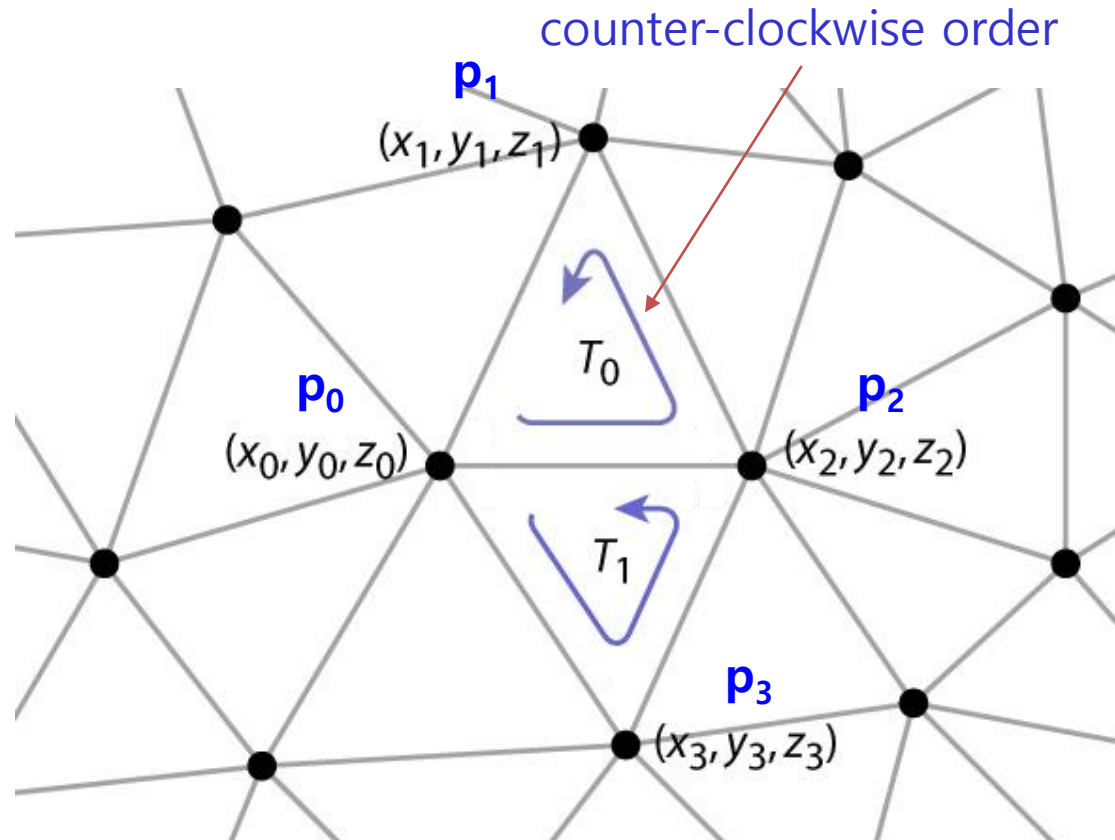
- The order of vertices defines the orientation of a polygon
- This determines the front and back face
 - CCW (counterclockwise) → front face (common convention)
 - CW (clockwise) → back face
- Vertex order must be stored consistently



Separate Triangles

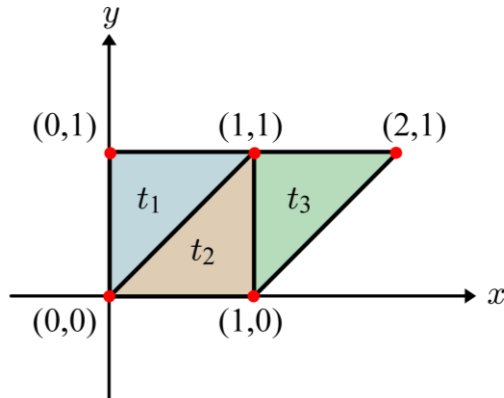
- Each triangle stores its own three vertices
- Vertices are duplicated across triangles

	[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
	$\vdots$	$\vdots$	$\vdots$



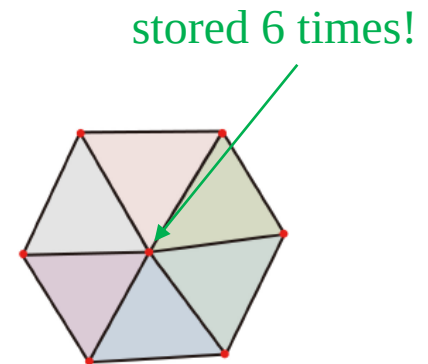
Separate Triangles - Limitations

- Wastes memory (duplicated vertices)
- Difficult to find neighboring triangles
 - No explicit connectivity information
 - e.g., To find neighbors of t_2 , you must look for vertices at the same position (zero-distance).



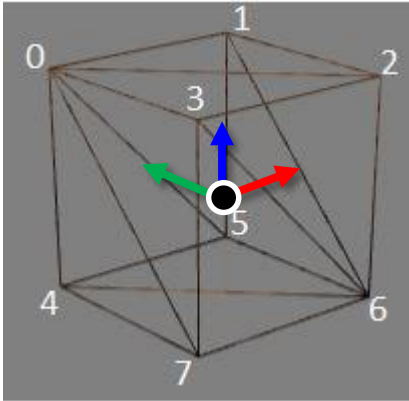
vertex buffer

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



(1,1) is stored 3 times!

Example: Cube

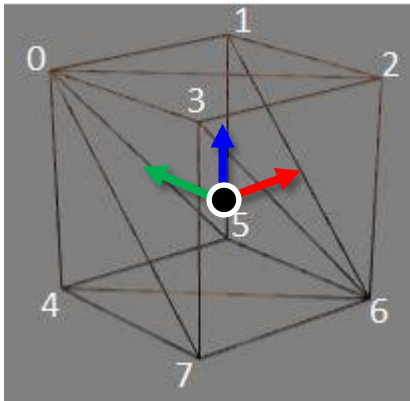


vertex index	position
0	$(-1, 1, 1)$
1	$(1, 1, 1)$
2	$(1, -1, 1)$
3	$(-1, -1, 1)$
4	$(-1, 1, -1)$
5	$(1, 1, -1)$
6	$(1, -1, -1)$
7	$(-1, -1, -1)$

Only 8 unique vertices

Cube with Separate Triangles

- 12 triangles → 36 vertices stored
- Vertices are duplicated across triangles

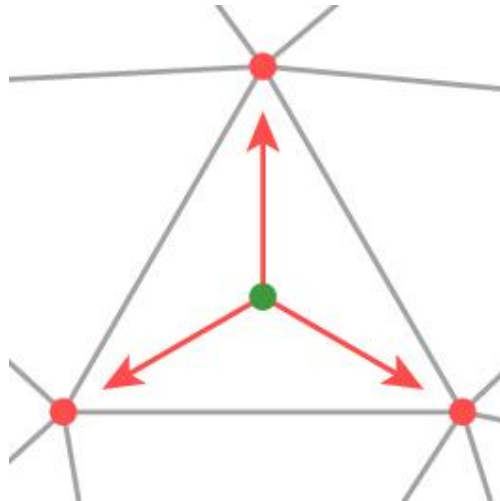


```
# vertex array
# triangle 0
-1 , 1 , 1, # v0
1 , -1 , 1, # v2
1 , 1 , 1, # v1
# triangle 1
-1 , 1 , 1, # v0
-1 , -1 , 1, # v3
1 , -1 , 1, # v2
# triangle 2
-1 , 1 , -1, # v4
1 , 1 , -1, # v5
1 , -1 , -1, # v6
# triangle 3
-1 , 1 , -1, # v4
1 , -1 , -1, # v6
-1 , -1 , -1, # v7
# triangle 4
-1 , 1 , 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 (no duplication)
- Each triangle “points to” its three vertices



Indexed Triangle Set

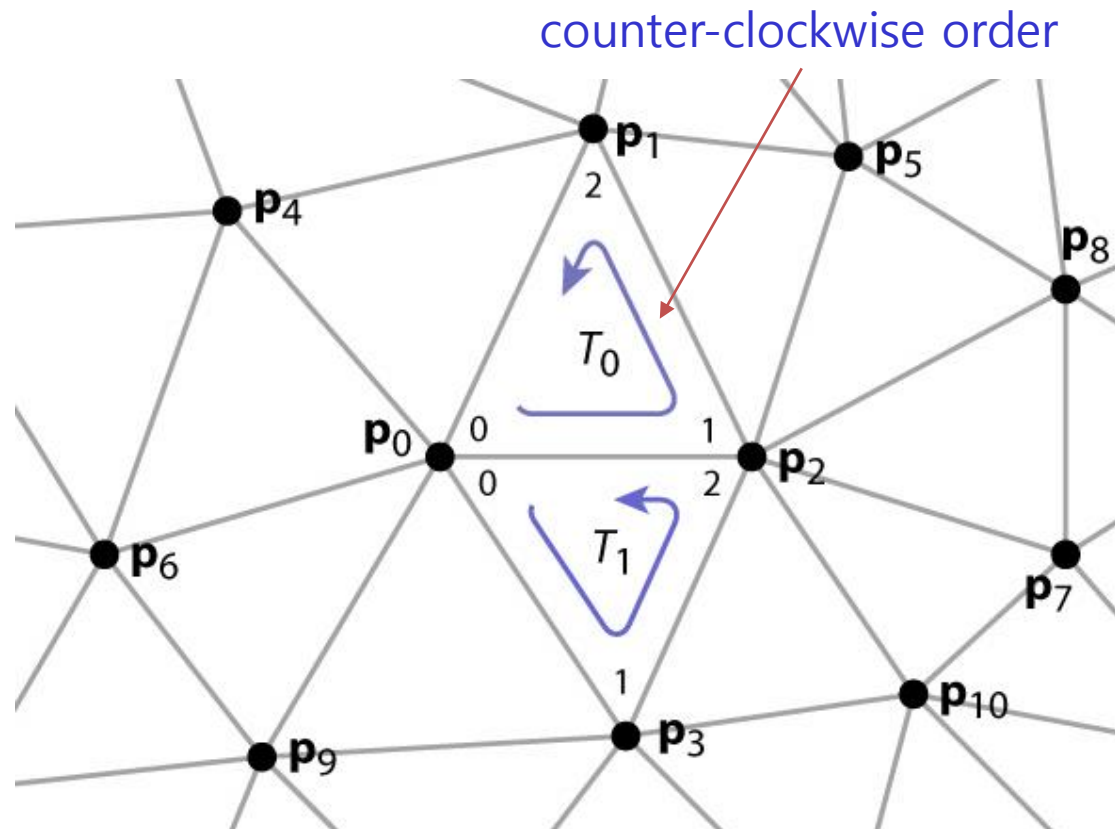
- Vertex array: unique vertices
- Index array: triangles defined by indices

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
	$\vdots$

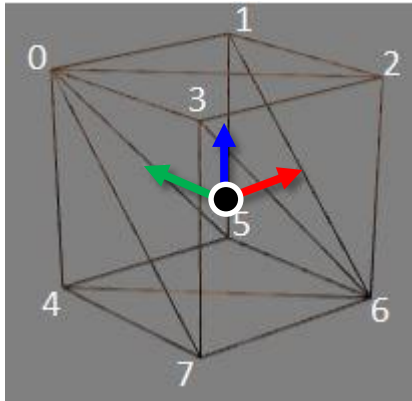


Indexed Triangle Set

- Memory efficient
 - Each vertex is stored only once.
- Finding neighbor triangles is at least well defined.
 - Neighbor triangles share vertex indices.
- Represents topology and geometry separately.
 - Topology: how vertices are connected
 - Geometry: vertex positions

Cube with Indexed Triangle Set

- 8 unique vertices
- 12 triangles → use indices
- → No duplicated vertices



```
# 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
```

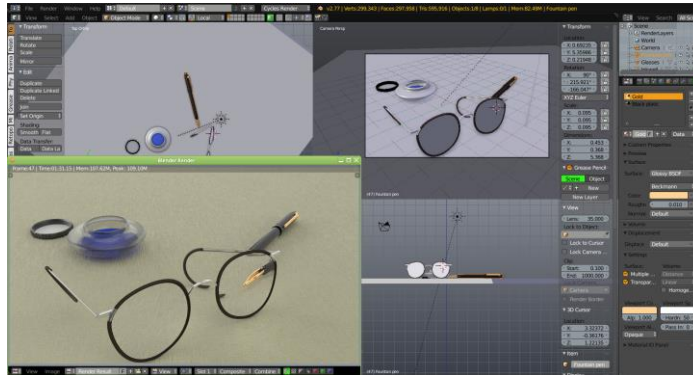
```
# index array
0,2,1, # t0
0,3,2, # t1
4,5,6, # t2
4,6,7, # t3
0,1,5, # t4
0,5,4, # t5
3,6,2, # t6
3,7,6, # t7
1,2,6, # t8
1,6,5, # t9
0,7,3, # t10
0,4,7, # t11
```

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. 2024123456: 4.0
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Creating Polygon Meshes

- Meshes are created using modeling tools (e.g., Blender, Maya)
- Mesh data is stored in files (*object file* or *model file*)
 - vertex positions
 - triangle indices
 - ...



Blender



Maya

- Applications load this data and render the mesh

3D Model File Formats

- Many formats exist
 - DXF, 3DS, VRML, OBJ, ...
- **OBJ – Wavefront OBJ format**
 - ASCII – human-readable
 - 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
...

# (Other formats include texture/normal indices)
#
vertex_position_index/texture_coordinates_index/vertex_normal_
index
f 6/4/1 3/5/3 7/6/5

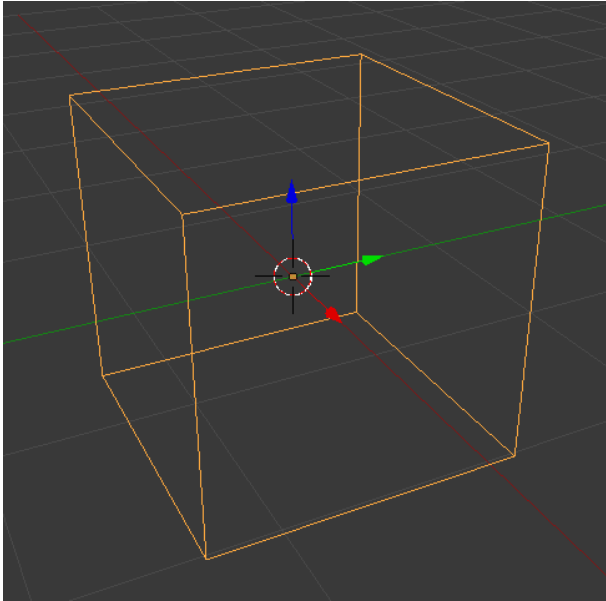
# vertex_position_index//vertex_normal_index
f 7//1 8//2 9//3
...

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

OBJ File Format

- Other supported features:
 - Polyline
 - `1 5 8 1 2 4 9 # connects vertices as a line`
 - Materials (color, texture, etc.)
 - `mtllib file.mtl # material defined in file.mtl`
 - `usemtl material1 # assign 'material1' to faces`
 - ...
- 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.